Trabajo de Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Control de un tiltrotor implementado en el autopiloto de código abierto *PX4*

Autor: Isidro Jesús Arias Sánchez

Tutores: Manuel Vargas Villanueva

Manuel Gil Ortega Linares

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo de Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Control de un tiltrotor implementado en el autopiloto de código abierto *PX4*

Autor:

Isidro Jesús Arias Sánchez

Tutores:

Manuel Vargas Villanueva

Profesor Titular

Manuel Gil Ortega Linares

Catedrático

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo de Fin de Grado: Control de un tiltrotor implementado en el autopiloto de código abierto *PX4*

Autor: Isidro Jesús Arias Sánchez
Tutores: Manuel Vargas Villanueva, Manuel Gil Ortega Linares

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Estos 4 años han sido mejores gracias al apoyo de algunas personas. Especialmente quiero agradecérselo a mis padres, Mercedes y José, a mi tía Antonia, a mi abuela Mercedes, a mi hermano Héctor, a Candi, mi novia, a mis amigos de Sevilla, Ángel, Carlos, David, Fernando, Jorge y Samuel, a los de Cádiz, Daniel, Juan Luís, Javier Mena, Javier Sanz, Pepe y Pedro, y por último y no menos importante, a mi tutores Manuel Vargas y Manuel Gil, y al resto de los profesores que me han sabido enseñar y de los que tanto he aprendido. Gracias a todos.

Isidro Arias Sánchez
Sevilla, 2019

Resumen

En este proyecto se va a desarrollar el estudio de un vehículo aéreo no tripulado que tiene dos hélices o rotores orientables, denominado tiltrotor, del ingles *tilt* que significa inclinar.

El fundamento de este trabajo es contribuir en el estudio de una nave convertible en la que los rotores, al ser inclinables, funcionan con las ventajas de dos modelos de aeronaves diferentes, una del tipo helicóptero que lo dota de alta maniobrabilidad, y otra de tipo aeroplano que le permite recorrer largas distancias.

Entendiendo la importancia de la simulación en la construcción de vehículos aéreos, se opta por la elección de dos herramientas diferentes en el ámbito de la simulación. Se trata de comprobar y testar las coincidencias entre ellas, de manera que se corrija y disminuya la posibilidad de errores en el proceso.

Por último, se añade al proyecto la fabricación de un tiltrotor para verificar los modelos utilizados en la simulación.

Abstract

The basis of this work is to contribute to the study of a convertible aerial vehicle in which the rotors, being tiltable, operate with the advantages of two different aircraft models, one of the helicopter type that gives it high maneuverability, and one of the airplane type that allows traveling long distances.

Understanding the importance of simulation in the design of aerial vehicles, we have chosen two different tools in the field of simulation. It is about verifying and testing the coincidences between them, so that it corrects and reduces the possibility of errors in the process.

Finally, a prototype will be built in order to verify the models used in the simulation.

Índice Abreviado

Índice

Notación

$\boldsymbol{P}$	Posición del vehículo en ejes inerciales
$\boldsymbol{P}_{sp}$	Posición deseada del vehículo en ejes inerciales
$\boldsymbol{V}$	Velocidad del vehículo en ejes inerciales
$\boldsymbol{V}_{sp}$	Velocidad deseada del vehículo en ejes inerciales
$\boldsymbol{F}_{NED}$	Fuerzas deseadas en el vehículo en ejes inerciales
ϕ_{sp}, θ_{sp}, ψ_{sp}	Orientación deseada expresada en angulos de euler
$\boldsymbol{q}_{sp}$	Orientación deseada expresada en cuaterniones
$\boldsymbol{\tau}_{sp}$	Pares deseados en ejes cuerpo
$\boldsymbol{\Omega}$	Velocidad angular en ejes cuerpo
$\boldsymbol{\Omega}_{sp}$	Velocidad angular deseada en ejes cuerpo
p, q, r	Componentes de Ω
R	Matriz de rotación del sistema de coordenadas del vehículo a los ejes inerciales
R_{sp}	Matriz de rotación de los ejes deseados del vehículo a los ejes inerciales

Acrónimos

UAV	Vehículo aéreo no tripulado
GCS	Estación de control terrestre
NED	Ejes norte, este y abajo
ESC	Controlador de velocidad electrónico (Electronic Speed Control)
SITL	Software-In-The-Loop
HITL	Hardware-In-The-Loop
ROS	Robot Operating System

1 Introducción

All models are wrong, but some are useful

George Box, 1976

El interés en los vehículos aéreos no tripulados UAV (*Unmanned Aerial Vehicle*) está en aumento. Dos grandes sectores en el mercado de los UAVs son el de consumo, como aquellos que se usan con fines de ocio, y el comercial, como los que son usados en los negocios o instituciones para realizar algún tipo trabajo. Un ejemplo de su aplicación se puede ver en una prueba piloto [1] que consiste en utilizar las torres de comunicaciones para albergar un UAV que se dirige de forma autónoma hasta la zona de un posible incendio, si recibe un aviso de sensores infrarrojos situados en al misma torre de comunicaciones.

Este último sector es el que se espera que experimente un mayor aumento y se predice que su tamaño se triplique en el año 2023 [2]. Su crecimiento se ve limitado por diversos factores entre los que destacan las regulaciones, como la necesidad de estar dentro del alcance visual del piloto, o la autonomía de las baterías. Esto último es más notorio en aquellos vehículos de ala rotativa como los helicópteros o quadrotors, ya que son mucho menos eficientes que los de ala fija como los aeroplanos. Sin embargo, tienen mayor maniobrabilidad que estos últimos puesto que, entre otras cosas, son capaces del despegue y aterrizaje en vertical.

Dadas las ventajas de cada uno de estos tipos nace la idea de *VTOL*, que son los aviones con capacidad de despegue vertical. Un caso de ellos son los tiltrotors, que son aquellos que poseen rotores (también llamados proprotores) que se inclinan. En [3] se hace una recopilación de los estudios sobre diferentes técnicas de control de estos vehículos, tanto de 2, 3 o 4 rotores. Para el caso de 2 rotores, en [4] y [5] se proponen distintos controladores.

En este trabajo se diseñará un controlador para el modo helicóptero de un tiltrotor y se implementará en un autopiloto de código abierto llamado *PX4*. Este es un proyecto comenzado en 2009 por la *Escuela Politécnica Federal de Zúrich* (ETH) que ofrece todo el software necesario para hacer que los vehículos aéreos, acuáticos o terrestres naveguen de forma autónoma.

Antes de diseñar un controlador y modificar el autopiloto para implementarlo, en el capítulo 2 se hará un análisis del mismo sin modificarlo. Una vez comprendido su funcionamiento, en el capitulo 3 se realizará el modelado matemático de un tiltrotor para posteriormente, en el capítulo 4, diseñar un controlador en *Matlab/Simulink* que se pretende que sea lo más parecido posible al que ya hay en *PX4*. Una vez hecho esto, en el capítulo 5 se utilizará un simulador llamado *Gazebo* junto con con el autopiloto ejecutado en el sistema operativo *Ubuntu*.

Figura 1.1 Prueba piloto de UAV antiincendios. Fuente [1] .

Por último, en el capítulo 6 se muestran las características y resultados de un tiltrotor que se ha fabricado para este trabajo.

2 Autopiloto *PX4*

En este este capítulo se va a describir el autopiloto en su forma original. En primer lugar se describirá de forma general, luego se enfocará en los controladores que incorpora y por último, en cómo se puede probar sin utilizar un vehículo real. Toda esta descripción se hará sobre una versión en concreto del autopiloto. Esta se encuentra alojada en el repositorio oficial de *PX4* y concretamente está en el commit 75bb3e9bac. De esta manera se podrá hacer referencia directa a las líneas de código del autopiloto, aunque se pretende que este trabajo sea lo suficientemente autocontenido como para no tener que visitar dicho repositorio.

2.1 Vista general de *PX4*

Debido a las necesidades de paralelismo en las tareas a realizar en un vehículo autónomo, se necesita un sistema operativo. Esto permite que disponiendo de un solo procesador, distintas tareas se estén ejecutando al mismo tiempo desde el punto de vista del programador, haciendo que el código reduzca su complejidad en comparación con el uso de un microcontrolador sin sistema operativo. Los sistemas operativos en los que se ha implementado son *Linux* y *NuttX*. Este último es un sistema operativo de tiempo real apto para microcontroladores de hasta tan solo 8 bits. Además, cumple la norma *POSIX*, por lo que requiere de menor aprendizaje si ya se conocen otros. Un ejemplo de hardware en el que se puede usar es el microcontrolador *STM32F427*, el cual tiene como núcleo un *Cortex-M4F*. Algunas de sus características son una frecuencia de reloj de 168MHz, una RAM (memoria dinámica) de 256 KB y una FLASH (memoria de programa) de 2MB. De microcontroladores como este, hace uso un proyecto de hardware abierto llamado *Pixhawk*. Este proyecto se encarga de diseñar estándares, en el que luego los fabricantes se basan para fabricar sus placas de circuito impreso. Un ejemplo de placa que cumple dicho estándar es *Pixhawk 2.1* (también conocido como *The Cube*). Esta incluye, además del microcontrolador antes mencionado, las siguientes características[1]:

- Coprocesador *STM32F100* para entradas y salidas.
- Conectores para las diferentes interfaces (UART, I2C, PPM...)
- 3 Unidades de Medición Inercial (IMU).
- Aislamiento de vibraciones para las IMUs.

[1] Para ver más características visitar el datasheet del fabricante en http://www.hex.aero/wp-content/uploads/2016/07/DRS_Pixhawk-2-17th-march-2016.pdf. Todos los esquemáticos están liberados y se encuentran en https://github.com/proficnc/The-Cube

Figura 2.1 *Pixhawk* 2.1 (*The Cube*). [Fuente: docs.px4.io].

En cuanto al software, tiene una estructura formada por módulos, los cuales en una implementación en *Linux* coinciden con hilos. En la figura 2.2 se muestran algunos de ellos en forma de rectángulo azul, como por ejemplo:

- *Position Controller*. Se encarga del control de posición y de velocidad. Además transforma los niveles de los sticks del la emisora a referencias de los controladores. Por último, también tiene la funcionalidad de generador de trayectorias.
- *Attitude & Rate Controller*. Se encarga del control de orientación y velocidad angular.
- *Output Driver*. Genera las señales de los actuadores en función de las fuerzas y pares deseados y gestiona el periférico PWM.
- *Autonomous Flight*. Recoge los waypoints de la memoria sd y los envía a *Position Controller*.
- *Position & Attitude Estimator*. Estimador de la posición y la orientación mediante un Filtro de Kalman Extendido (EKF). Está documentado en la carpeta Firmware/src/lib/ecl/EKF/documentation.
- *MAVLink*. Protocolo de comunicación con la GCS.
- *FastRTPS*. Permite la comunicación subscriptor-publicador entre dispositivos diferentes.

Para la comunicación entre los módulos, existe un bus de mensajes con una arquitectura de publicador-subscriptor (parecido al que se usa en ROS). Esto permite una mayor independencia entre los módulos (para que un módulo se comunique con otro, solo tienen que conocer la estructura del mensaje) y mayor escalabilidad (los distintos módulos no tienen que ser ejecutados en el mismo computador). Algunos de los mensajes[2] usados se representan mediante flechas en la figura 2.2.

2.2 Modos de vuelo

El autopiloto presenta varios modos de vuelo, automáticos y manuales, que pueden ser activados o no dependiendo de ciertas condiciones como la disponibilidad de algunos sensores. Desde el punto de vista del controlador, los modos de vuelo afectan en qué tipo de referencias hay disponibles (posición, velocidad, fuerza...) y con ello cuáles de los controladores están en funcionamiento o cuáles se desactivan. Los modos de vuelo que se van a comentar son *position*, *stabilized*, *acrobatic* y *auto*

2.2.1 Modo *position*

Es este modo se generan las siguientes referencias:

- V_{x_aux}, V_{y_aux}. Velocidades de referencia expresadas en un sistema de coordenadas auxiliar. Este resulta de rotar el sistema de coordenadas inercial con el ángulo ψ, que es uno de los ángulos de euler que expresan la rotación del vehículo. Estos ejes coincidirán con los del vehículo si los ángulos ϕ y θ fuesen nulos. Estas dos velocidades se generan a partir de los canales de la emisora radio-control. La relación entre la posición de las palancas y la velocidad es tal que una posición extrema de la palanca, produzca la velocidad máxima permitida dada por el parámetro *MPC_VEL_MANUAL*. Estas referencias serán convertidas [3] a ejes inerciales que es como las acepta el controlador.
- V_{z_sp}. Velocidad de referencia en el eje vertical. Una velocidad positiva indica un movimiento hacia el suelo, ya que las coordenadas están en NED.
- P_{x_sp}, P_{y_sp} y P_{z_sp}. Cuando la velocidad de referencia es nula se comienza a generar, además de la velocidad, una posición de referencia constante coincidente con el lugar en el que se situaba el vehículo en el primer momento en el que la velocidad deseada era cero.
- $\dot{\psi}_{sp}$. La velocidad de referencia del ángulo yaw.

Además, antes de pasar la referencia de velocidad al controlador, a esta se le limita la pendiente para disminuir las sobreoscilaciones del seguimiento. La pendiente máxima está dada por el parámetro *MPC_ACC_HOR*[4] e indica la aceleración máxima que se desea en el seguimiento de velocidad.

[2] La lista completa de los mensajes utilizados, junto con su descripción, se encuentra en la carpeta *Firmware/msg/*

[3] dicha transformación se realiza en la función *_rotateIntoHeadingFrame* del archivo Firmware/src/lib/FlightTasks/tasks/ManualAltitude/FlightTaskManualAltitude.cpp

[4] El parámetro para la aceleración y desaceleración máxima no es el mismo, el de esta última denomina *MPC_ACC_HOR_MAX*

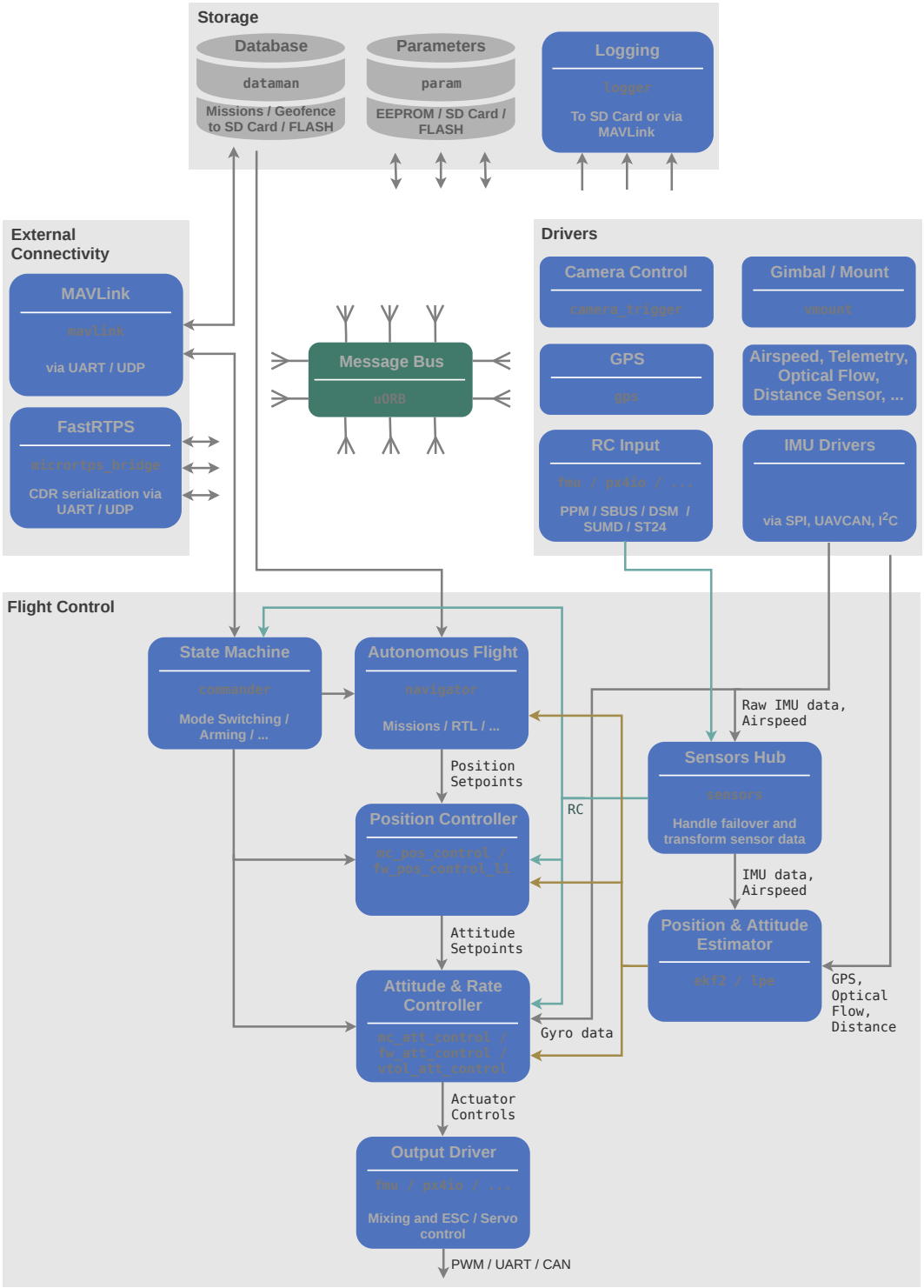

Figura 2.2 Vista general de PX4. [Fuente: dev.px4.io].

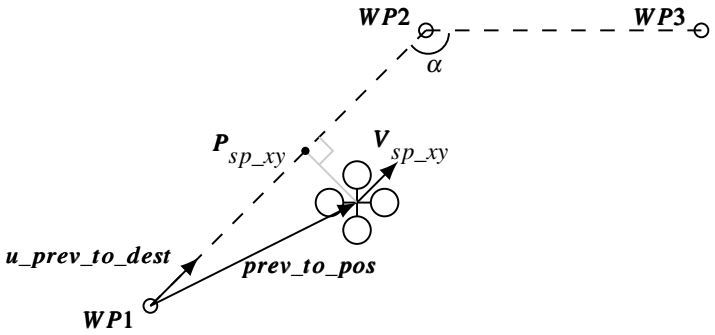

Figura 2.3 Referencias de posición y velocidad en el seguimiento a recta.

2.2.2 Modo *stabilized*

En este modo las referencias generadas por la emisora rc son[5]:

- ϕ_{sp} y θ_{sp} [rad]. Ángulos de euler *roll* y *pitch* deseados.
- Derivada temporal del ángulo yaw ($\dot{\psi}_{sp}$) [rad/s]. Limitado por *MPC_MAN_Y_MAX* y es proporcional a la palanca de yaw.
- ψ_{sp} Álgulo yaw deseado [rad]. Este se obtiene al integrar con respecto al tiempo la posición de la palanca de yaw.
- F_z. Fuerza deseada en el eje z del vehículo.

2.2.3 Modo *acrobatic*

Se generan las siguientes referencias a partir de la emisora rc:

- p, q, r, velocidades angulares en los ejes cuerpo. Los valores máximos los establecen los parámetros *MC_ACRO_R_MAX*, *MC_ACRO_P_MAX* y *MC_ACRO_Y_MAX*.
- F_z, fuerza deseada en el eje z del vehículo.

Cuando se empieza a diseñar un controlador, será este el modo que primero se utilice, ya que puede funcionar con independencia de los demás controladores que están en niveles superiores.

Las referencias que se generan no son directamente proporcionales al nivel de las palancas del mando, sino que se obtienen a través de una función exponencial que depende de dos parámetros: *MC_ACRO_EXPO* y *MC_ACRO_SUPEXPO* [6]. Esta característica puede ser una ayuda a la hora de diseñar un controlador, ya que si le asignamos al segundo parámetro un valor cercano a 1, es posible generar un escalón en la señal de control si se realiza movimiento rápido de la palanca del mando. De esta forma se pueden medir los tiempos de subida con independencia de la rapidez con la que se mueva la palanca.

2.2.4 Modo *auto*

El objetivo de este modo es alcanzar una serie de puntos en el espacio (*waypoints*) en un orden especificado y en línea recta. Este modo aporta al controlador tanto velocidad como posición y ángulo yaw de referencia. Estos se generan de la siguiente manera[7]:

- $P_{sp\ x}$ $P_{sp\ y}$. La posición se escoge como el punto más cercano de la recta que une el *waypoint* objetivo con el *waypoint* anterior. Para calcularlo se definen la siguientes variables auxiliares que se muestran en la figura 2.3:

[5] La genración de las referencias a partir de los niveles de los sticks del mando está implementado en la función *generate_attitude_setpoint* del archivo *src/modules/mc_att_control/mc_att_control_main.cpp*

[6] La función que hay implementada y los efectos de los dos parámetros puede verse en https://www.desmos.com/calculator/yty5kgurmc. En el código, la obtención de lar referencia p, q y r se encuentra en las líneas 929-934 del archivo *Firmware/src/modules/mc_att_control/mc_att_control_main.cpp*

[7] Tanto la generación de las referencias de velocidad como las de posición, están implementadas en la función _generateXYsetpoints del archivo Firmware/src/lib/FlightTasks/tasks/AutoLine/FlightTaskAutoLine.cpp

- Posición del vehículo:

$$\boldsymbol{P} = \begin{bmatrix} P_x \\ P_y \end{bmatrix} \tag{2.1}$$

- Posición del *waypoint* objetivo:

$$\boldsymbol{WP2} = \begin{bmatrix} WP2_x \\ WP2_y \end{bmatrix} \tag{2.2}$$

- Posición del *waypoint* origen:

$$\boldsymbol{WP1} = \begin{bmatrix} WP1_x \\ WP1_y \end{bmatrix} \tag{2.3}$$

- Vector unitario en la dirección de la recta que une $\boldsymbol{WP1}$ y $\boldsymbol{WP2}$:

$$\boldsymbol{u\_prev\_to\_dest} = \frac{\boldsymbol{WP1} - \boldsymbol{WP2}}{\|\boldsymbol{WP1} - \boldsymbol{WP2}\|} \tag{2.4}$$

- Vector desde el *waypoint* origen hasta el vehículo:

$$\boldsymbol{prev\_to\_pos} = \boldsymbol{P} - \boldsymbol{WP1} \tag{2.5}$$

Finalmente, se calcula la posición más cercana al vehículo como la suma de $\boldsymbol{WP1}$ y la proyección de $\boldsymbol{prev\_to\_pos}$ en la dirección de $\boldsymbol{u\_prev\_to\_dest}$. Se hace uso de la operación producto escalar:

$$\boldsymbol{P}_{xy\_sp} = \boldsymbol{WP1} + \boldsymbol{u\_prev\_to\_dest}(\boldsymbol{prev\_to\_pos} \cdot \boldsymbol{u\_prev\_to\_dest}) \tag{2.6}$$

- $V_{sp\ x}$ y $V_{sp\ y}$. La velocidad de referencia siempre tiene la dirección de la recta que une el waypoint anterior con el waypoint objetivo ($\boldsymbol{u\_prev\_to\_dest}$). Para el cálculo de la magnitud del vector se definen las siguientes variables:
  - *speed_at_target*. Velocidad que debe de tener el vehículo cuando se llegue al waypoint. Esta depende del ángulo α (ver figura 2.3) y será menor cuanto más alejado esté este ángulo de 180°.
  - *MPC_XY_CRUISE*. Parámetro que indica la velocidad máxima en modo *auto*.
  - *target_threshold*. Distancia al objetivo a la cual se empieza a desacelerar. Por defecto es igual a 1.5·*MPC_XY_CRUISE*.
  - *acceptance_radius*. Si la distancia del vehículo al *waypoint* objetivo es menor que esta cantidad, se considera que ya ha sido alcanzado. En ese momento el punto objetivo pasa ser el siguiente de la misión.

  Para empezar, la magnitud de $\boldsymbol{V}_{xy\_sp}$ va aumentado con pendiente constante (aceleración constante) hasta que se iguale al parámetro *MPC_XY_CRUISE*. Una vez que la distancia del *UAV* al objetivo es menor que *target_threshold*, la velocidad disminuirá proporcionalmente a la distancia hasta el *waypoint* menos *acceptance_radius*, de manera de que cuando se llegue a él, la velocidad de referencia sea *speed_at_target*.
- $P_{sp\ z}$, $V_{sp\ z}$. Para controlar la altura[8], se generan posiciones y velocidades de referencia en función de la distancia en vertical del UAV al waypoint objetivo (d_z), y de la distancia al waypoint anterior (d_{z_prev}), las cuales se representan en la figura 2.4. Tiene distintas de etapas y estará en una u otra dependiendo de las siguientes condiciones:
  1. Si d_{z_prev} < *target_threshold*:
     Se aumenta la velocidad con aceleración constante igual al parámetro *MPC_ACC_UP_MAX*

[8] Implementado en la función _generateAltitudeSetpoints del archivo Firmware/src/lib/FlightTasks/tasks/AutoLine/FlightTaskAutoLine.cpp

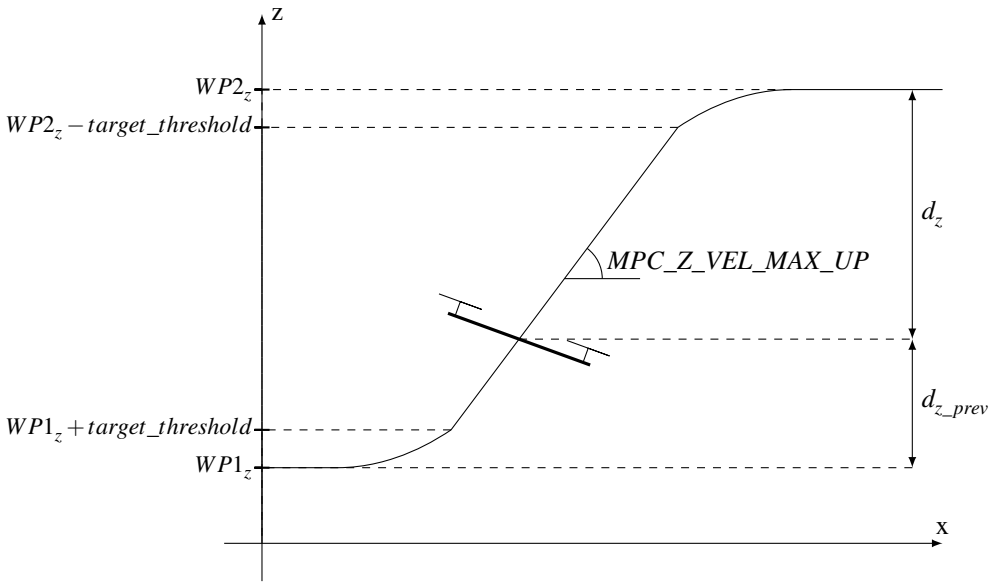

Figura 2.4 Generación de la velocidad deseada en el eje z. Suponiendo que la velocidad en el eje x es constante y el seguimiento de la altura deseada es perfecto.

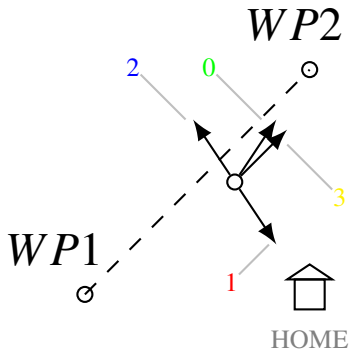

Figura 2.5 Dirección de yaw en función del parámetro *MPC_YAW_MODE*.

2. Si $d_{z\_prev} > target\_threshold$ y además $d_z > target\_threshold$:
   Se persigue la altura con velocidad constante igual al parámetro *MPC_Z_VEL_MAX_UP*
3. Si $d_z < target\_threshold$:
   Se va reduciendo la velocidad proporcional a d_z.
4. Si d_z=0:
   La velocidad de referencia es cero y se activa el control de posición para el eje z.

- ψ_{sp}. El ángulo yaw se genera de diferentes maneras dependiendo del valor del parámetro *MPC_YAW_MODE* (ver figura 2.5):
  - 0: Hacia el waypoint.
  - 1: Hacia el lugar de partida (*Home*).
  - 2: En dirección opuesta a *Home*.
  - 3: Paralelo a la trayectoria.

2.3 Bloques del control

Como puede observarse en la figura 2.6, en este autopiloto existe un control en cascada. Puede verse que existen 4 contralores, un bloque de conversión de fuerzas a orientación y un control *mixer*, que convierte de

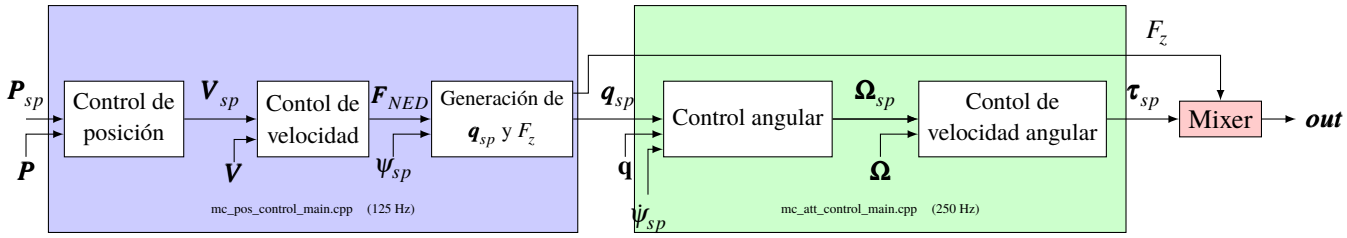

Figura 2.6 Diagrama de bloques del control.

fuerzas y pares a señales de actuación. Las variables de referencia se denotan con el subíndice $_{sp}$ (setpoint). Las variables **P**, **V** son la posición y velocidad estimadas en ejes inerciales y **q** representa la orientación estimada. Mientras que estas estimaciones provienen de un filtro de Kalman extendido, la velocidad angular ($\boldsymbol{\Omega}$) se recoge directamente del giroscopio. $\boldsymbol{F}_{NED}$ es la fuerza deseada en ejes inerciales. En cuanto al control *mixer*, las variables de entrada son la fuerza deseada en el eje z del vehículo (F_z) y los pares deseados también en el sistema de referencia del vehículo ($\boldsymbol{\tau}_{sp}$); su salida es un vector de señales de actuación normalizadas (**out**). El control en cascada permite que se pueda elegir entre el seguimiento de referencias de distintos tipos. Además permite un diseño paso por paso pudiendo desactivar cuando se desee los controladores de alto nivel, para dar referencias de bajo nivel, y así poder sintonizar por separado los diferentes controladores. Cada uno de los bloques que lo componen se van a ir describiendo esta sección.

2.3.1 Control de posición

Este controlador solo tiene termino proporcional. Su objetivo es hacer que la posición $\boldsymbol{P}$ siga a la referencia $\boldsymbol{P}_{sp}$. La salida de este controlador es la velocidad de referencia ($\boldsymbol{V}_{sp}$) que será utilizada por el control de velocidad. Además del término proporcional, a la salida se le suma otra velocidad de referencia ,$V_{sp\ aux}$, que no es generada por este controlador, sino por el generador de trayectorias que se explicó en la subsección 2.2.4. Este y el siguiente controlador se ejecutan cada vez que se recibe una nueva posición estimada, que ocurre a una tasa de 125 hz (8.0ms)[9].

$$\boldsymbol{V}_{sp} = \begin{bmatrix} v_{x\ sp} \\ v_{y\ sp} \\ v_{z\ sp} \end{bmatrix} = \boldsymbol{K_p} *^{[10]} (\boldsymbol{P}_{sp} - \boldsymbol{P}) + \boldsymbol{V}_{sp\ aux} \qquad [\text{m/s}] \quad (2.7)$$

Si se cumple la desigualdad

$$\left\| \begin{bmatrix} v_{x\ sp} \\ v_{y\ sp} \end{bmatrix} \right\| < MPC\_XY\_VEL\_MAX \quad (2.8)$$

la referencia de velocidad se pasa directamente al controlador de velocidad, si no, hay que transformarla para que esté dentro de los límites. En tal caso la transformación mantiene la dirección del vector y es la siguiente[11]:

$$\left. \begin{bmatrix} v_{x\ sp} \\ v_{y\ sp} \end{bmatrix} \right|_{sat} = \frac{\begin{bmatrix} v_{x\ sp} \\ v_{y\ sp} \end{bmatrix}}{\left\| \begin{bmatrix} v_{x\ sp} \\ v_{y\ sp} \end{bmatrix} \right\|} \cdot MPC\_XY\_VEL\_MAX \quad (2.9)$$

[9] Este tiempo podría ser variable (por ejemplo si existe demasiada carga computacional) así que en cada iteración se lee el intervalo de tiempo que ha pasado. Este dato es utilizado para el cálculo de la derivada e integral del error en el controlador de velocidad

[10] La operación "*" denota una multiplicación componente a componente entre dos vectores

[11] En realidad la relación es más complicada cuando hay presente tanto referencia de posición como de velocidad. La saturación está implementada y comentada en la función constrainXY del archivo Firmware/src/modules/mc_pos_control/Utility/ControlMath.cpp. En ella se realiza una satuación que prioriza la velocidad que depende de la posición deseada ($\boldsymbol{K_p} * (\boldsymbol{P}_{sp} - \boldsymbol{P})$), frente a la velocidad de referencia ($\boldsymbol{V}_{sp\ aux}$).

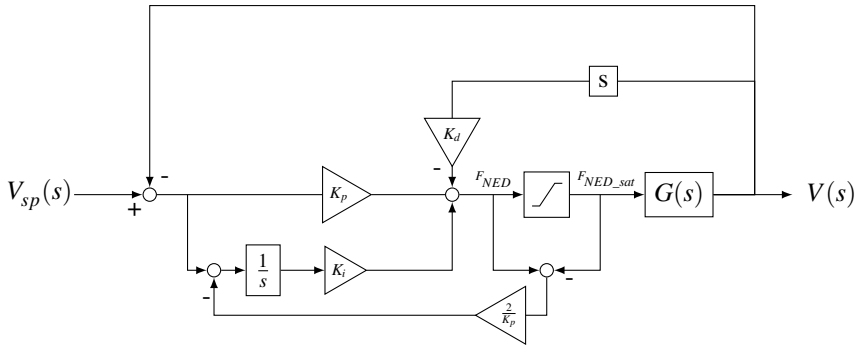

Figura 2.7 Diagrama de bloques del control de la velocidad para los ejes xy.

2.3.2 Control de velocidad

Este controlador es un PID. Existen dos controladores separados, uno para la velocidad del eje z y otro para los ejes x e y. En cuanto al primero, su ley de control es la siguiente:

$$F_{z\_NED} = K_p(V_{z\_sp} - V_z) - K_d\dot{V}_z + K_i e_{int} - MPC\_THR\_HOVER \qquad [\text{N}] \quad (2.10)$$

Donde:

- *MPC_THR_HOVER*. Parámetro que indica la fuerza la fuerza necesaria en el eje z para vencer la gravedad.
- e_{int} es la integral del error. Su actualización se produce en cada ciclo a no ser que la señal de control sature ($F_{z\_NED} < -MPC\_THR\_MAX$ o $F_{z\_NED} > 0$), en cuyo caso solo se actualizará si favorece la desaturación (línea 268 del archivo *PositionControl.cpp*). Además, la integral se limitará para que cumpla la siguiente condición:

$$|\, e_{int} \,|\, K_i < MPC\_THR\_MAX \quad (2.11)$$

- K_d es la ganancia de la derivada, la cual solo se aplica a la derivada de la medida de la velocidad, no al error como suele ser habitual, por esa razón tiene signo negativo.

Para los ejes xy la ley de control es la siguiente y se ilustra en la figura 2.7:

$$\boldsymbol{F}_{xy\_NED} = K_p(\boldsymbol{V}_{xy\_sp} - \boldsymbol{V}_{xy}) - K_d\dot{\boldsymbol{V}}_{xy} + K_i\boldsymbol{e}_{int} \qquad [\text{N}] \quad (2.12)$$

Hay que destacar los siguientes puntos:

1. Para la acción derivativa, es necesario estimar la derivada de la velocidad. Esto se realiza en dos pasos: primero se realiza una derivada discreta a la velocidad, y luego se filtra mediante un filtro de paso bajas con un solo polo. [12]
2. El módulo de la señal de control $\boldsymbol{F}_{xy\_NED}$ se puede saturar por dos razones (ver figura 2.8):
   a) Máxima inclinación permitida. Se establece mediante el parámetro *MPC_MAN_TILT_MAX*. Para abreviar, aquí se denotará como β_{max}

$$F_{max\_1} = F_{z\_NED}\ \tan\beta_{max} \quad (2.13)$$

   b) Máximo empuje que se puede generar (T_{max}).

$$F_{max\_2} = \sqrt{T_{max}^2 - F_{z\_NED}} \quad (2.14)$$

[12] La implementación de la derivada discreta está en el archivo *BlockDerivative.cpp* y el de el filtro en *BlockLowPass.cpp*. La frecuencia de corte es igual al parámetro *MPC_VELD_LP*.

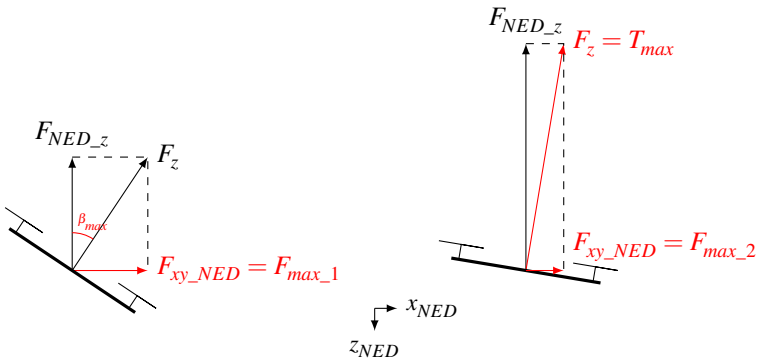

Figura 2.8 Formas de saturar F_{xy_NED}.

La señal de control será saturada con el mínimo de los dos valores anteriores:

$$\boldsymbol{F}_{NED\_sat} = min(F_{max\_1}, F_{max\_2}, F_{xy\_NED}) \frac{\boldsymbol{F}_{xy\_NED}}{\|\boldsymbol{F}_{xy\_NED}\|} \tag{2.15}$$

3. $\boldsymbol{e}_{int}$ es la integral del error y tiene efecto anti-windup. En la iteración n, se calcula en función del valor en la iteración anterior $(e_{int}(n-1))$ y del tiempo transcurrido (Δt):

$$\boldsymbol{e}_{int}(n) = \boldsymbol{e}_{int}(n-1) + \Delta t \left((\boldsymbol{V}_{xy\_sp} - \boldsymbol{V}_{xy}) - (\boldsymbol{F}_{xy\_NED} - \boldsymbol{F}_{NED\_sat}) \frac{2}{K_p} \right) \tag{2.16}$$

4. La ganancias K_p, K_i y K_d son escalares, y corresponden a los siguientes parámetros *MPC_XY_VEL_P*, *MPC_XY_VEL_I* y *MPC_XY_VEL_D*.

2.3.3 Generación de orientación y fuerza

En este bloque[13] se hace una conversión de la fuerza deseada generada por el control de velocidad ($\boldsymbol{F}_{NED}$). A partir de esta se calcula una fuerza en el eje z del vehículo (F_z) y una orientación deseada expresada en cuaternios ($\boldsymbol{q}_{sp}$). Esta se calcula de tal manera que su eje z sea coincidente con $\boldsymbol{F}_{NED}$ y además tenga en cuenta el ángulo ψ_{sp}. Para ello se definen las siguientes variables:

1. $\boldsymbol{F}_{NED}$. Vector de 3 dimensiones que indica la fuerza de referencia en ejes inerciales generada por el control de velocidad.
2. ψ_{sp}. Ángulo yaw de referencia.
3. ***body_x***, ***body_y*** , ***body_z***. Base de vectores unitarios del sistema de referencia del cuerpo medidos en el eje inercial.

Los multirrotores normalmente solo son capaces de producir un empuje en dirección negativa de su eje z, por lo tanto se hace coincidir la dirección de ***body_z*** con el vector dado $\boldsymbol{F}_{NED}$.

$$\boldsymbol{body\_z} = -\frac{\boldsymbol{F}_{NED}}{\|\boldsymbol{F}_{NED}\|} \tag{2.17}$$

Para hallar los otros dos ejes restantes se tendrá en cuenta el ángulo ψ_{sp}. ***body_x*** que situará de tal manera que sea perpendicular a ***body_z*** (ya que debe de resultar una base ortonormal) y de forma que su proyección

[13] Implementado en la función thrustToAttitude del archivo Firmware/src/modules/mc_pos_control/Utility/ControlMath.cpp

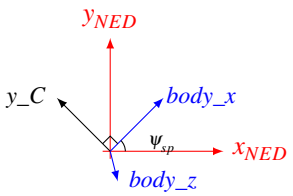

Figura 2.9 Proyección de los vectores en el plano xy.

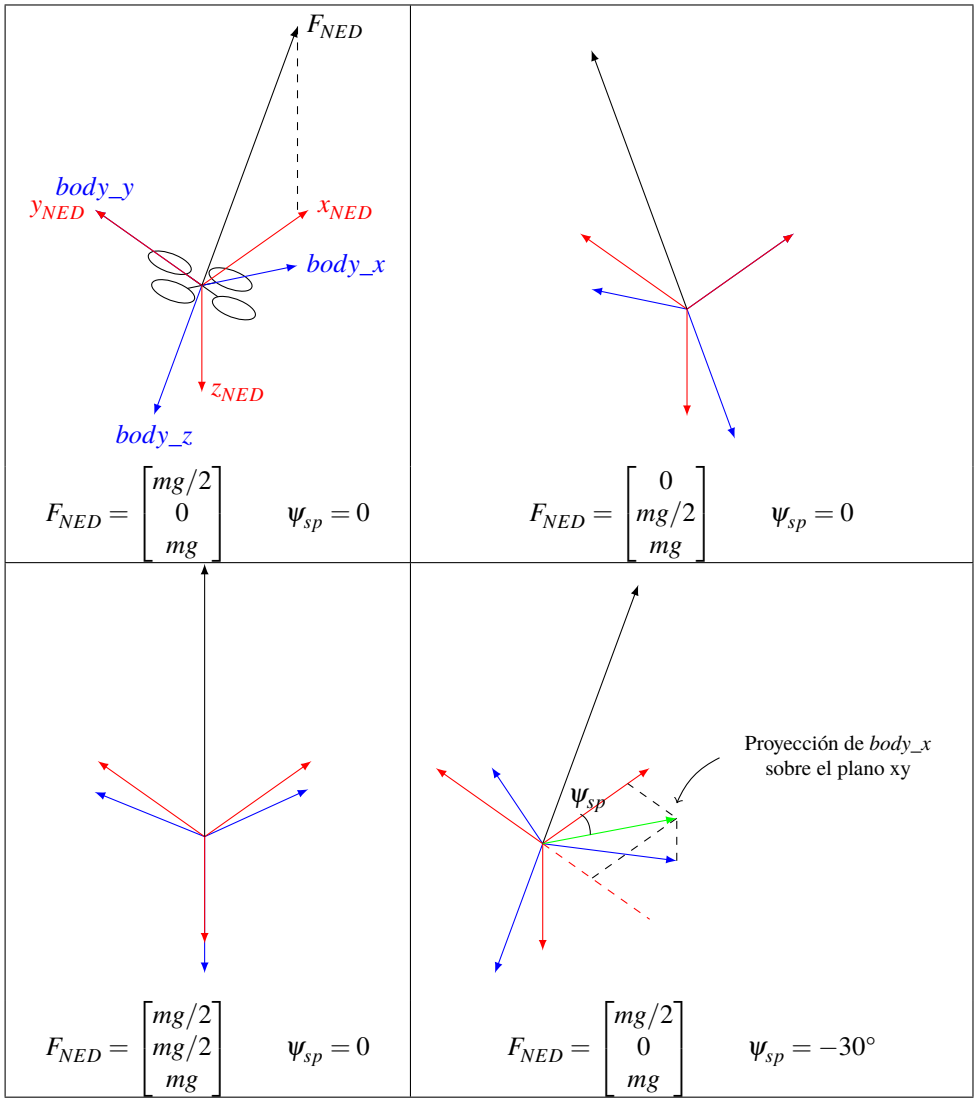

Figura 2.10 Rotación.

en el plano XY inercial tenga un ángulo igual a ψ_{sp}. Para ello se define la variable auxiliar ***y_C*** como el vector unitario en el plano XY inercial con ángulo ψ_{sp} girado 90 grados (ver figura 2.9).

$$\mathbf{y\_C} = \begin{bmatrix} cos(\pi/2) & -sen(\pi/2) & 0 \\ sen(\pi/2) & cos(\pi/2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\psi_{sp}) \\ sen(\psi_{sp}) \\ 0 \end{bmatrix} = \begin{bmatrix} -sen(\psi_{sp}) \\ cos(\psi_{sp}) \\ 0 \end{bmatrix} \tag{2.18}$$

$$\mathbf{body\_x} = \frac{\mathbf{body\_z} \times \mathbf{y\_C}}{\|\mathbf{body\_z} \times \mathbf{y\_C}\|} \tag{2.19}$$

Para terminar de formar la base queda calcular **body_y**:

$$\mathbf{body\_y} = \mathbf{body\_z} \times \mathbf{body\_x} \tag{2.20}$$

A partir de esto tres vectores obtenidos, se calcula la matriz de rotación (R_{sp}) . Esta servirá para expresar la rotación en forma de cuaterniones ($\boldsymbol{q}_{sp}$) y ángulos de euler[14]. Es la matriz de rotación de los ejes cuerpo deseados a los inerciales.

$$R_{sp} = \begin{bmatrix} \mathbf{body\_x} & \mathbf{body\_y} & \mathbf{body\_z} \end{bmatrix} \tag{2.21}$$

Por último, la fuerza deseada en el eje z del vehículo será el módulo de $\boldsymbol{F}_{NED}$:

$$F_z = \|\boldsymbol{F}_{NED}\| \tag{2.22}$$

2.3.4 Control de ángulo

Una explicación detallada de este controlador se puede ver en [6]. En esta sección unicamente se hará un resumen de lo desarrollado en dicho artículo y se cambiará su notación a una más parecida a la existente en el código [15] del autopiloto. La ley de control se basa en cuaterniones y es la siguiente.

$$\boldsymbol{\Omega}_{sp} = 2\boldsymbol{K}_p * \boldsymbol{q}_{e,1:3}\ sign(q_{e,0}) + \boldsymbol{\Omega}_{yaw} \quad \text{[rad/s]} \tag{2.23}$$

Donde:

1. $\boldsymbol{K}_p$. Es el vector de ganancias del control. Se corresponde con los parámetros de la siguiente manera:

$$\boldsymbol{K}_p = \begin{bmatrix} MC\_ROLL\_P \\ MC\_PITCH\_P \\ \frac{MC\_ROLL\_P + MC\_PITCH\_P}{2} \end{bmatrix} \tag{2.24}$$

2. $\boldsymbol{q_e} = \boldsymbol{q}^{-1} \cdot \boldsymbol{q}_{sp,red}$. Este es un producto de cuaternios donde $\boldsymbol{q}$ es la orientación estimada y $\boldsymbol{q}_{sp,red}$ es la orientación deseada reducida. $\boldsymbol{q}_{sp,red}$ se calcula a partir de la orientación deseada ($\boldsymbol{q}_{sp}$), que proviene del bloque anterior, y de una variable auxiliar *yaw_w*. Esta última se obtiene a partir del parámetro *MC_YAW_P* de la siguiente manera y está limitada al rango de 0 a 1:

$$yaw\_w = \frac{MC\_YAW\_P}{(MC\_ROLL\_P + MC\_PITCH\_P)/2} \tag{2.25}$$

   Esta variable funciona como peso del control sobre el eje z. Cuando vale 0, no se produce ninguna rotación sobre él, y cuando vale 1 se cumple que $\boldsymbol{q}_{sp,red} = \boldsymbol{q}_{sp}$. Para hallar $\boldsymbol{q}_{sp,red}$ con valores intermedios de *yaw_w*, se define una orientación auxiliar $\boldsymbol{q}_{sp,red\_max}$ que se calcula de la siguiente manera:

$$\boldsymbol{q}_{sp,red\_max} = \boldsymbol{q} \begin{bmatrix} cos\left(\frac{\alpha}{2}\right) \\ sen\left(\frac{\alpha}{2}\right)\left(\frac{\boldsymbol{z} \times \boldsymbol{z}_{sp}}{\|\boldsymbol{z} \times \boldsymbol{z}_{sp}\|}\right) \end{bmatrix} \tag{2.26}$$

   Donde $\boldsymbol{z}$ se refiere al vector z de la orientación actual y $\boldsymbol{z}_{sp}$ al de la orientación deseada (en la anterior subsección se ha denominado $\boldsymbol{body\_z}$). Estos se calculan a partir de las orientaciones expresadas en cuaternios $\boldsymbol{q}$ y $\boldsymbol{q}_{sp}$ respectivamente, convirtiendolas a matrices de rotación y extrayendo su tercera columna. Por otra parte, α es el ángulo comprendido entre los vectores $\boldsymbol{z}$ y $\boldsymbol{z}_{sp}$ y se calcula de la siguiente manera:

$$\alpha = acos\left(\boldsymbol{z}^T \cdot \boldsymbol{z}_{sp}\right) \tag{2.27}$$

   De esta manera la orientación $\boldsymbol{q}_{sp,red\_max}$ corresponde con la orientación en la que se produce la mínima rotación hasta $\boldsymbol{q}$ que tenga su eje z apuntando hacia $\boldsymbol{z}_{sp}$. Esto se traduce en que el eje de rotación que lleva $\boldsymbol{q}$ hasta $\boldsymbol{q}_{sp,red\_max}$ pertenece al plano *xy* de $\boldsymbol{q}$ [16].

[14] Los ángulos de euler no se utilizarán para el control, solo para mostrar la orientación de una manera intuitiva al usuario

[15] Dicho código se encuentra en la función *control_attitude* del archivo *Firmware/src/modules/mc_att_control/mc_att_control_main.cpp*.

[16] Hay que aclarar que esto no provoca que $\boldsymbol{q}_{sp,red\_max}$ no tenga en cuenta el ángulo yaw de $\boldsymbol{q}_{sp}$ (ψ_{sp}). Ya que $\boldsymbol{q}_{sp,red\_max}$ apunta en la dirección de ψ_{sp}. En otras palabras, ψ_{sp} determina la direccion de la inclinación pero no afecta en el giro sobre el eje z.

Además, se define la variable $\boldsymbol{q}_{mix}$ como la rotación de $\boldsymbol{q}_{sp,red\_max}$ a $\boldsymbol{q}_{sp}$:

$$\boldsymbol{q}_{mix} = \boldsymbol{q}^{-1}_{sp,red\_max} \cdot \boldsymbol{q}_{sp} \tag{2.28}$$

Puesto que la única rotación entre estas dos orientaciones se produce en el eje z, $\boldsymbol{q}_{mix}$ tendrá la siguiente forma:

$$\boldsymbol{q}_{mix} = \begin{bmatrix} cos\left(\frac{\alpha_{mix}}{2}\right) \\ 0 \\ 0 \\ \sin\left(\frac{\alpha_{mix}}{2}\right) \end{bmatrix} \tag{2.29}$$

Se reordena la equación 2.28:

$$\boldsymbol{q}_{sp} = \boldsymbol{q}_{sp,red\_max} \cdot \boldsymbol{q}_{mix} \tag{2.30}$$

Para hacer que $\boldsymbol{q}_{sp,red}$ pueda tomar cualquier valor entre $\boldsymbol{q}_{sp,red\_max}$ (no haber rotación sobre el eje z) y $\boldsymbol{q}_{sp}$ (sí hay rotación para buscar que ψ llegue a ψ_{sp}) en función de *yaw_w*, se define dicha variable de la siguiente manera:

$$\boldsymbol{q}_{sp,red} = \boldsymbol{q}_{sp,red\_max} \cdot \begin{bmatrix} cos\left(yaw\_w\frac{\alpha_{mix}}{2}\right) \\ 0 \\ 0 \\ \sin\left(yaw\_w\frac{\alpha_{mix}}{2}\right) \end{bmatrix} \tag{2.31}$$

3. $\boldsymbol{\Omega}_{yaw} = R^{-1}\begin{bmatrix} 0 \\ 0 \\ \dot{\psi}_{sp} \end{bmatrix}$. Donde R es la matriz de rotación de los ejes cuerpo a los inerciales y $\dot{\psi}_{sp}$ es la velocidad deseada del ángulo yaw. Por definición, el ángulo ψ es una rotación en el eje z inercial, por ello se sitúa en la tercera componente del vector que multiplicará la matriz R^{-1}.

2.3.5 Control de velocidad angular

Este controlador genera pares deseados ($\boldsymbol{\tau}_{sp}$) a partir de errores en la velocidad angular ($\boldsymbol{\Omega}$). Se ejecuta cada vez que se recibe una nueva lectura del giroscopio, que ocurre a 250 Hz (4 ms). La ley de control es la siguiente:

$$\boldsymbol{\tau}_{sp} = \boldsymbol{K}_p * (\boldsymbol{\Omega}_{sp} - \boldsymbol{\Omega}) - \boldsymbol{K}_d * (\dot{\boldsymbol{\Omega}}) + \boldsymbol{K}_{FF} * \boldsymbol{\Omega}_{sp} + \boldsymbol{K}_i * \boldsymbol{e}_{int} \quad \text{[N m]} \tag{2.32}$$

Donde:

- $\boldsymbol{K}_d$ es la ganancia del término derivativo. Corresponde con los parámetros *MC_ROLLRATE_D* ,*MC_PITCHRATE_D* y *MC_YAWRATE_D*.
- $\dot{\boldsymbol{\Omega}} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}$ es la aceleración angular estimada. Para calcularla, primero se filtra $\boldsymbol{\Omega}$ (resultando $\boldsymbol{\Omega}_{filt}$) y luego se le realiza una derivada discreta. Para el caso del eje x la expresión es la siguiente:

$$\dot{p}(n) = \frac{p_{filt}(n) - p_{filt}(n-1)}{\Delta t} \tag{2.33}$$

El filtro es uno de paso bajas y de tipo biquad en la forma directa 2. La ecuación en diferencias es la siguiente:

$$p_{filt}(n) = b_0 w(n) + b_1 w(n-1) + b_2 w(n-2) \tag{2.34}$$

$$w(n) = p(n) - a_1 p(n-1) - a_2 p(n-2) \tag{2.35}$$

En el dominio de Z se expresa de la siguiente manera:

$$H(z) = \frac{p_{filt}(z)}{p(z)} = \frac{b_0 + b_1 z^{-1} + b_2^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.36}$$

El diseño del filtro se realiza mediante la conversión de un filtro de Butterworth a discreto. La función de transferencia segundo orden es la siguiente:

$$G(s) = \frac{1}{s^2 + \sqrt{2}s + 1} \tag{2.37}$$

Para convertirlo a discreto se utilizará la transformación bilineal, que transforma los polos en el dominio z, al dominio s de la siguiente manera:

$$s = K^{-1}\frac{1 - z^{-1}}{1 + z^{-1}} \tag{2.38}$$

$$K = \tan\left(\frac{\omega_c T_m}{2}\right) \tag{2.39}$$

Donde ω_c es la frecuencia de corte deseada, que se obtiene del parámetro *MC_DTERM_CUTOFF* y T_m es el tiempo de muestreo del filtro que es el mismo que el de este controldor.

Si se sustituye en la función de transferencia en $G(s)$, se obtiene el filtro en discreto:

$$H(z) = \frac{K^2 + 2K^2 z^{-1} + K^2 z^{-2}}{K^2 + \sqrt{2}K + 1 \ + \ 2(K^2 - 1)z^{-1} \ + \ (K^2 - K\sqrt{2} + 1)z^{-2}} \tag{2.40}$$

Para reordenarlo se define la siguiente variable intermedia:

$$DE = K^2 + \sqrt{2}K + 1 \tag{2.41}$$

$$H(z) = \frac{K^2 + 2K^2 z^{-1} + K^2 z^{-2}}{DE \ + \ 2(K^2 - 1)z^{-1} \ + \ (K^2 - \sqrt{2} + 1)z^{-2}} = \frac{\frac{K^2}{DE} + \frac{2K^2}{DE}z^{-1} + \frac{K^2}{DE}z^{-2}}{1 \ + \ 2\frac{K^2 - 1}{DE}z^{-1} \ + \ \frac{K^2 - K\sqrt{2} + 1}{DE}z^{-2}} \tag{2.42}$$

Por comparación con la ecuación 2.36 se obtienen los coeficientes:

$$b_0 = \frac{K^2}{DE} \tag{2.43}$$

$$b_1 = 2 \cdot \frac{K^2}{DE} \tag{2.44}$$

$$b_2 = \frac{K^2}{DE} \tag{2.45}$$

$$a_1 = 2 \cdot \frac{K^2 - 1}{DE} \tag{2.46}$$

$$a_2 = \frac{1 - K\sqrt{2} + K^2}{DE} \tag{2.47}$$

- $\boldsymbol{K}_{FF}$ es un la ganancia feed-forward, la cual multiplica a la referencia de velocidad angular. Esta se usa para desacoplar el diseño del rechazo a perturbaciones y el del seguimiento a referencia. Equivale a los parámetros *MC_ROLLRATE_FF* ,*MC_PITCHRATE_FF* y *MC_YAWRATE_FF*.
- $\boldsymbol{K}_i$ es la ganancia del término integral. Hay 3 condiciones en las que la integral del error no se actualiza:
  - Cuando el vehículo está aterrizado.
  - Si llega a un valor límite (ver figura 2.11) establecido por los parámetros *MC_RR_INT_LIM*, *MC_PR_INT_LIM* y *MC_YR_INT_LIM* que afectan a los ejes x, y y z respectivamente.
  - Si algún actuador está saturado la integral solo se actualiza si es para favorecer su desaturación[17].

[17] Para ello el módulo *mixer* publica un mensaje de tipo *multirotor_motor_limits* en el que se indica, para cada uno de los pares deseados, si su incremento o decremento provocaría que la saturación de los motores aumentase.

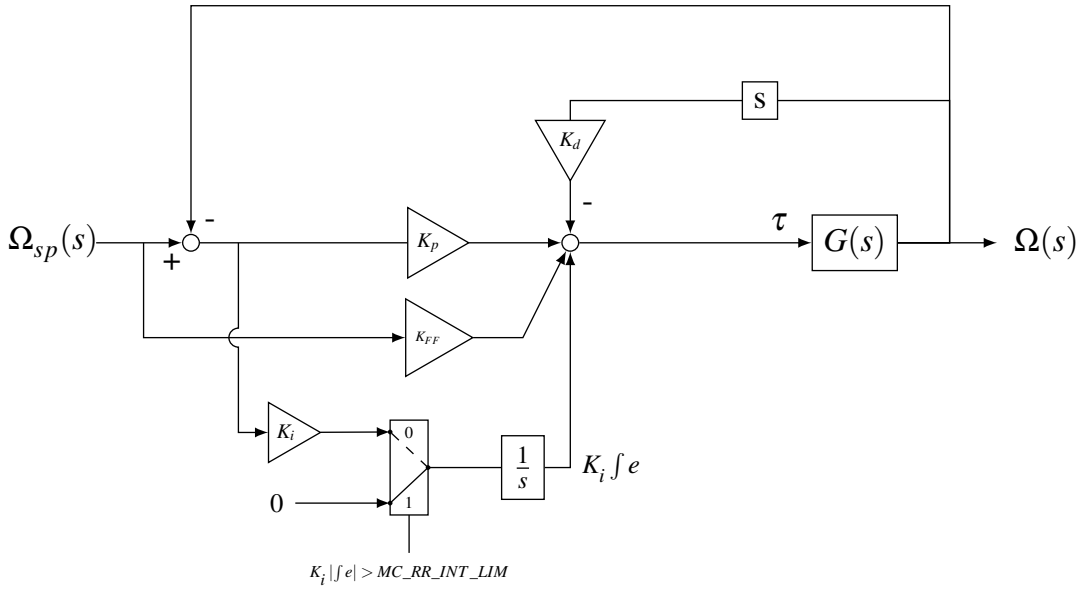

Figura 2.11 Diagrama de bloques del control de velocidad angular.

2.3.6 Control *mixer*

Se le llama control *mixer* a la obtención de señales de control de los actuadores, a partir de las fuerzas y los pares deseados. En este autopiloto hay disponibles 3 categorías: *mixer* simple, *mixer* de multirotor y *mixer* de helicóptero. El *mixer* de multirotor genera las señales de actuación a partir de multiplicar una matriz y un vector de fuerzas y pares deseados. Hay varias matrices definidas y estas son generadas automáticamente a partir de un archivo de parámetros[18]. En este archivo hay que especificar ciertos parámetros sobre la ubicación de los rotores y las constantes de los motores. Un ejemplo de este archivo se muestra en el siguiente código:

Código 2.3.1: Fichero de parámetros para crear un *mixer*

```
# Generic Quadcopter in + configuration

[info]
key = "4+"
description = "Generic Quadcopter in + configuration"

[rotor_default]
direction = "CW"
axis      = [0.0, 0.0, -1.0]
Ct        = 1.0
Cm        = 0.05

[[rotors]]
name      = "right"
position  = [0.0, 1.0, 0.0]
direction = "CCW"

[[rotors]]
name      = "left"
position  = [0.0, -1.0, 0.0]
direction = "CCW"

[[rotors]]
name     = "front"
position = [1.0, 0.0, 0.0]

[[rotors]]
name     = "rear"
```

[18] Esta tarea la realiza el programa *px_generate_mixers.py* ubicado en Firmware/src/lib/mixer/geometries/tools/ y los archivos se generan en /Firmware/build/px4_sitl_default/src/lib/mixer. Este no se ejecuta en tiempo de ejecución, sino en tiempo de compilación.

```
29  position = [-1.0, 0.0, 0.0]
```

Los parámetros que aparecen son los siguientes:

- C_T, la constante de empuje.
- C_m, la constante de par de arrastre.
- **axis**, la dirección del eje de rotación
- **position**, la posición del rotor con respecto al centro de masas.
- *direction*, el sentido de giro. Puede ser horario "CW" o antihorario "CCW".

En primer lugar, se calculan las fuerzas y pares que genera cada rotor i. Para ello es necesario definir una variable intermedia ***out*** que representa la señal de los actuadores normalizada. La definición es:

$$out_i = \left(\frac{\omega_i}{\omega_{max}}\right)^2 \tag{2.48}$$

Donde ω_i es la velocidad del rotor i en radianes por segundo y ω_{max} es la velocidad máxima en esa misma unidad. A partir de esta se pueden definir los pares y fuerzas:

- La fuerza de empuje:

$$\mathbf{F_i} = c_{Ti}\left(\frac{\omega_i}{\omega_{max}}\right)^2 \mathbf{axis_i} = c_{Ti}\, out_i\, \mathbf{axis_i} \tag{2.49}$$

- El par de reacción:

$$\boldsymbol{\tau}_{di} = -c_{mi}\left(\frac{\omega_i}{\omega_{max}}\right)^2 dir \cdot \mathbf{axis} \tag{2.50}$$

$$dir = \begin{cases} 1 & si\ direction = "CW" \\ -1 & si\ direction = "CCW" \end{cases} \tag{2.51}$$

- El par generado al aplicar una fuerza en un punto que no es el centro de masas:

$$\boldsymbol{\tau}_{Fi} = c_{Ti}\left(\frac{\omega_i}{\omega_{max}}\right)^2 \mathbf{position_i} \times \mathbf{axis_i} \tag{2.52}$$

Hay que notar que los parámetros c_T y c_m no tienen el significado habitual. Por convención estas relacionan el cuadrado de la velocidad angular con el empuje y el par que generan. Sin embargo, en este autopiloto se ha implementado de otra manera, y el significado que adquiere es del empuje o par máximo que es capaz de generar el rotor. Esto se puede comprobar al sustituir ω_i por ω_{max} en la versión escalar de la ecuación 2.49.

$$F_{max} = c_T\ \left(\frac{\omega_{max}}{\omega_{max}}\right)^2 = c_T \cdot 1 \tag{2.53}$$

Estas igualdades se expresan en forma matricial,

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \\ F_z \end{bmatrix} = A \begin{bmatrix} out_1 \\ out_2 \\ out_3 \\ out_4 \end{bmatrix} \tag{2.54}$$

donde:

$$A = \begin{bmatrix} -c_{m1} dir_1 \cdot \mathbf{axis_1} + c_{T1} \mathbf{position_1} \times \mathbf{axis_1} & -c_{m2} dir_2 \cdot \mathbf{axis_2} + c_{T2} \mathbf{position_2} \times \mathbf{axis_2} & \cdots \\ F_{z1} & F_{z2} & \cdots \end{bmatrix} \tag{2.55}$$

Una vez que se obtiene A, se le realiza la pseudoinversa para despejar las señales de los actuadores. En el caso del quadrotor, la matrix A es invertible y par tanto su pseudoinversa coincide con su inversa.La razón de

utilizar la pseudoinversa es para el caso de UAVs con un número diferente de 4 rotores ya que la matriz A ya no sería cuadrada.

$$\mathbf{out} = A^{+} \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \\ F_z \end{bmatrix} \tag{2.56}$$

Hay que tener en cuenta que los motores suelen ir conectados a un variador de velocidad (ESC), y este recibe del microcontrolador una velocidad de giro deseada en forma de PWM. La señal de PWM suele tener un ancho de pulso que va desde $1000\mu s$ (motor en régimen de giro mínimo) hasta $2000\mu s$ (máxima velocidad). Teniendo en cuenta esto se puede obtener el ancho de pulso en microsegundos de la señal PWM:

$$PWM_i = \sqrt{out_i}1000 + 1000 \qquad [\mu s] \tag{2.57}$$

2.4 Estimación de los estados

Una parte muy importante del desempeño de los controladores depende de conocer los estados del vehículo como la orientación o la velocidad. Algunos de ellos no pueden estimarse directamente con la medida de los sensores, como por ejemplo suele ser la orientación, y además, siempre existe un ruido en la medida que provoca incertidumbre en la estimación. Para ello están los estimadores de estados como el filtro de Kalman, los cuales pueden aprovecharse de que existen sensores que aportan información sobre los mismos estados (por ejemplo el magnetómetro y el giróscopo), de manera que cada uno favorece la estimación. Además, aportan información sobre la incertidumbre que se tiene en el conocimiento de los estados, que matemáticamente se expresa en forma de matriz de covarianzas.

2.4.1 Introducción al EKF

Para la estimación de los estados y de la matriz de covarianzas se usa un filtro de Kalman extendido. Su diferencia con respecto a un filtro de Kalman es que puede ser aplicado a sistemas no lineales como el que se muestra a continuación:

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{2.58}$$

$$\boldsymbol{z}_{k+1} = h(\boldsymbol{x}_k) + \boldsymbol{v}_k \tag{2.59}$$

Donde:

- $\boldsymbol{x}_k$ es el vector de estados de tamaño n.
- $\boldsymbol{u}_k$ vector de entradas.
- f es la función de perdición. Puede ser no lineal y tener múltiples entradas y salidas.
- $\boldsymbol{w}_k$ Error del modelo de predicción. Este es un ruido gaussiano aditivo con media nula y matriz de covarianzas Q_k.
- $\boldsymbol{z}_k$ es el vector de observaciones de tamaño m.
- h es la función de observación. Puede ser no lineal y tener múltiples entradas y salidas.
- $\boldsymbol{v}_k$ Vector de ruidos en las medidas. Debe de ser gaussiano aditivo con media nula. Su matriz de covarianzas es R_k.

Este es un algoritmo recursivo cuyas entradas son los estados estimados ($\boldsymbol{x}_{k-1}$) y la matriz de covarianzas (P_{k-1}) ambas de la iteración anterior y sus salidas son $\boldsymbol{x}_k$ y P_k. Está compuesto por los siguientes pasos:

1. Predicción de los estados

$$\hat{\boldsymbol{x}}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{2.60}$$

2. Predicción de la matriz de covarianzas

$$\hat{P}_k = F_k P_{k-1} F_k^T + Q_k \tag{2.61}$$

Donde F_k es el jacobiano del modelo de predicción evaluado en los estados anteriores:

$$F_k = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{k-1}, \boldsymbol{u}_k} \tag{2.62}$$

3. Cálculo de la innovación

$$\boldsymbol{y}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_k) \tag{2.63}$$

4. Covarianza de la innovación

$$\hat{S}_k = H_k \hat{P}_k H_k^T + R_k \tag{2.64}$$

Donde H_k es el jacobiano del modelo de observación evaluado en los estados predichos:

$$H_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_k} \tag{2.65}$$

5. Ganancia de Kalman

$$K_f = \hat{P}_k H_k S_k^{-1} \tag{2.66}$$

6. Actualización de los estados

$$\boldsymbol{x}_k = \hat{\boldsymbol{x}}_k + K_f \boldsymbol{y}_k \tag{2.67}$$

7. Actualización de la matriz de covarianzas

$$P_k = \hat{P}_k + K_f H_k \hat{P}_k \tag{2.68}$$

Antes de comenzar una nueva iteración, los estados y la matriz de coverianza actuales se convierten en las pasadas ($\boldsymbol{x}_{k-1} = \boldsymbol{x}_k$ y $P_{k-1} = P_k$).

2.4.2 Estados

En *PX4* se ha elegido los siguientes estados:

- Orientación expresada en cuaterniones

$$\boldsymbol{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{2.69}$$

- Velocidad en ejes inerciales

$$\boldsymbol{V} = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} \quad \text{[m/s]} \tag{2.70}$$

- Posición en ejes inerciales

$$\boldsymbol{P} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \quad \text{[m]} \tag{2.71}$$

- Error en la medida de la variación de ángulo que se produce al integrar el error de offset del giróscopo:

$$\boldsymbol{\Delta Ang}_{bias} = \boldsymbol{\omega}_{bias} \Delta t \quad \text{[rad]} \tag{2.72}$$

Es equivalente al error de offset en el giróscopo ($\boldsymbol{\omega}_{bias}$) multiplicado por el tiempo de muestreo ($\Delta\, t$). El error de offset se tendrá en cuenta, además, en el acelerómetro y en el magnetómetro. Esto permite cumplir el requisito del filtro de Kalman de que el ruido de las medidas tengan media nula.

- Medida de la variación de velocidad al integrar el error de offset del acelerómetro:

$$\boldsymbol{\Delta Vel}_{bias} = \boldsymbol{a}_{bias}\Delta t \quad \text{[m/s]} \tag{2.73}$$

- $\boldsymbol{M}_{NED}$ [gauss]. Campo magnético terrestre expresado en ejes inerciales (NED). Este no siempre tiene la dirección del norte geográfico debido a la declinación magnética.
- $\boldsymbol{M}_{bias}$ [gauss]. Error de offset del magnetómetro.

Si se agrupan, forman un vector de 22 estados:

$$\boldsymbol{x}_k = \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{V} \\ \boldsymbol{P} \\ \boldsymbol{\Delta Ang}_{bias} \\ \boldsymbol{\Delta Vel}_{bias} \\ \boldsymbol{M}_{NED} \\ \boldsymbol{M}_{bias} \end{bmatrix} \tag{2.74}$$

2.4.3 Modelo de predicción

La ecuaciones de predicción son discretas y se utilizarán las medidas del giróscopo y del acelerómetro. De esta manera se evita tener que conocer el modelo dinámico del vehículo y con ello no es necesario conocer ni las inercias ni las señales de actuación.

1. Predicción de la orientación

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k \cdot \boldsymbol{\Delta q} \tag{2.75}$$

Donde:

$$\boldsymbol{\Delta q} = \begin{bmatrix} 1 \\ \frac{\Delta Ang_{truth\_x}}{2} \\ \frac{\Delta Ang_{truth\_y}}{2} \\ \frac{\Delta Ang_{truth\_z}}{2} \end{bmatrix} \tag{2.76}$$

$\boldsymbol{\Delta Ang}_{truth}$ es la lectura del giróscopo ($\boldsymbol{\omega}$) corregida:

$$\boldsymbol{\Delta Ang}_{truth} = \boldsymbol{\omega}\Delta t - \boldsymbol{\Delta Ang}_{bias} \tag{2.77}$$

2. Predicción de la velocidad

$$\boldsymbol{V}_{k+1} = \boldsymbol{V}_k + R^{B\to I}\boldsymbol{\Delta Vel}_{truth} + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \Delta t \tag{2.78}$$

Donde $\boldsymbol{\Delta Vel}_{truth}$ es la lectura del acelerómetro ($\boldsymbol{a}$) corregida:

$$\boldsymbol{\Delta Vel}_{truth} = \boldsymbol{a}\,\Delta t - \boldsymbol{\Delta Vel}_{bias} \tag{2.79}$$

3. Predicción de la posición

$$\boldsymbol{P}_{k+1} = \boldsymbol{P}_k + \boldsymbol{V}_k\Delta t \tag{2.80}$$

4. Modelo estático para el resto de los estados:

$$\begin{bmatrix} \boldsymbol{\Delta}_{ang\_bias} \\ \boldsymbol{\Delta}_{vel\_bias} \\ \boldsymbol{M}_{NED} \\ \boldsymbol{M}_{bias} \end{bmatrix}_{k+1} = \begin{bmatrix} \boldsymbol{\Delta}_{ang\_bias} \\ \boldsymbol{\Delta}_{vel\_bias} \\ \boldsymbol{M}_{NED} \\ \boldsymbol{M}_{bias} \end{bmatrix}_{k} \tag{2.81}$$

El ruido en la predicción (Q) está ocasionado por el ruido del acelerómetro y del giróscopo. Para ello primero se obtiene la dependencia de los estados con las medidas del acelerómetro y giróscopo realizando el siguiente jacobiano [19] .

$$G = \frac{\partial f}{\partial \boldsymbol{a}, \boldsymbol{\omega}} \tag{2.82}$$

la matriz de covarianzas del acelerómetro y giróscopo se obtiene a partir de los parámetros:

$$\begin{aligned} distMatrix = diag\big(\big[&EKF2\_ACC\_NOISE^2 \quad EKF2\_ACC\_NOISE^2 \quad EKF2\_ACC\_NOISE^2 \\ &EKF2\_GYR\_NOISE^2 \quad EKF2\_GYR\_NOISE^2 \quad EKF2\_GYR\_NOISE^2\big]\big) \end{aligned} \tag{2.83}$$

Siendo *diag* una función de *Matlab* que convierte un vector en una matriz diagonal. Por último, se utiliza el jacobiano para calcular Q:

$$Q = G\, distMatrix\, G^T \tag{2.84}$$

2.4.4 Actualización de los estados

La actualización de los estados engloban los pasos desde 2.63 hasta 2.68. En *PX4* se realiza fusión secuencial, que consiste en aplicar las medidas de los sensores en pasos separados, en lugar de aplicar todas en una sola operación matricial como ocurre en el paso indicado en la ecuación 2.67. Esto permite que el conjunto de sensores que utiliza el filtro no sea fijo y se pueda modificar en tiempo de vuelo. Además, se reduce el coste computacional.

PX4 es compatible con una gran variedad de sensores como los de flujo óptico, sensores láser para medir distancias o sensores para medir la velocidad del viento. Todos ellos tienen una manera especial para añadirlos al filtro, pero los más esenciales son los siguientes:

- Magnetómetro. $\boldsymbol{M}_{mag}$

$$\boldsymbol{M}_{mag} = R^{I\to B}\boldsymbol{M}_{NED} + \boldsymbol{M}_{bias} \qquad [\text{Gauss}] \tag{2.85}$$

$$H_{mag} = \frac{\partial \boldsymbol{M}_{mag}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial x_1}{\partial M_{mag\_x}} & \cdots & \frac{\partial x_{22}}{\partial M_{mag\_x}} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_1}{\partial M_{mag\_z}} & \cdots & \frac{\partial x_{22}}{\partial M_{mag\_z}} \end{bmatrix} \tag{2.86}$$

$$R_{mag} = \begin{bmatrix} EKF2\_MAG\_NOISE^2 & 0 & 0 \\ 0 & EKF2\_MAG\_NOISE^2 & 0 \\ 0 & 0 & EKF2\_MAG\_NOISE^2 \end{bmatrix} \tag{2.87}$$

- Velocidad GPS. $\boldsymbol{V}_{GPS}$

$$H_{V\_GPS} = \begin{bmatrix} 0 & \dots & 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & \underbrace{0}_{5:7} & \underbrace{0}_{5:7} & \underbrace{1}_{5:7} & 0 & \dots & 0 \end{bmatrix} \tag{2.88}$$

$$R_{V\_GPS} = \begin{bmatrix} EKF2\_GPS\_V\_NOISE^2 & 0 & 0 \\ 0 & EKF2\_GPS\_V\_NOISE^2 & 0 \\ 0 & 0 & (1.5 \cdot EKF2\_GPS\_V\_NOISE)^2 \end{bmatrix} \tag{2.89}$$

- Posición GPS. $\boldsymbol{P}_{GPS}$

$$H_{GPS} = \begin{bmatrix} 0 & \dots & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & \underbrace{0}_{8,9} & \underbrace{1}_{8,9} & 0 & \dots & 0 \end{bmatrix} \tag{2.90}$$

[19] Este y otros jacobianos se obtienen con ayuda del *Symbolic Math Toolbox* de *Matlab* en Firmware/src/lib/EKF/matlab/scripts/Inertial Nav EKF/GenerateNavFilterEquations.m:

$$R_{GPS} = \begin{bmatrix} EKF2\_GPS\_P\_NOISE^2 & 0 \\ 0 & EKF2\_GPS\_P\_NOISE^2 \end{bmatrix} \tag{2.91}$$

- Barómetro. h_{bar}:

$$H_{baro} = \begin{bmatrix} 0 & \dots & 0 & \underbrace{1}_{10} & 0 & \dots & 0 \end{bmatrix} \tag{2.92}$$

$$R_{baro} = EKF2\_BARO\_NOISE^2 \tag{2.93}$$

Antes de actualizar los estados con alguna medida, se comprueba que esta no sea muy diferente de lo predicho. Para ello evalua lo siguiente:

$$y_k < innovation\_gate \sqrt{\hat{S}_k} \tag{2.94}$$

Donde y_k es la innovación $\left(y_k = z_k - h(x_k)\right)$ de una medida cualquiera, $\hat{S}_k$ es la varianza de la innovación de esa medida y *innovation_gate* es un parámetro definido por el usuario. Algunos de estos son *EKF2_BARO_GATE*, *EKF2_GPS_P_GATE* y *EKF2_MAG_GATE* correspondientes al barómetro, GPS y magnetómetro respectivamente. Equivale a cuántas veces mayor puede ser la innovación a la desviación típica de la medida. Si no se cumple, se descarta la medida y se envía un mensaje de peligro.

2.4.5 inicialización

En la primera iteración del filtro, los estados y la matriz de covarianzas anteriores se desconocen, por lo que es necesario darles valor.

2.4.5.1 Estados

Una parte de la orientación (ángulos de euler ϕ y θ) se inicializa en base a la lectura del acelerómetro suponiendo velocidad nula:

$$\theta = asin\left(\frac{a_z}{\left\|\begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T\right\|}\right) \tag{2.95}$$

$$\phi = atan\left(\frac{a_y}{a_z}\right) \tag{2.96}$$

A partir de estos dos ángulos se obtiene el estado $\boldsymbol{q}$ (orientación expresada en cuaternios). Para el otro ángulo que queda se utiliza el magnetómetro. Primero se rota las lecturas del magnetómetro $\boldsymbol{M}_{med}$ utilizando $\boldsymbol{q}$, transformando las medidas al plano horizontal ($\boldsymbol{M}_{med\_rot}$). A partir de este y de la declinación magnética [20], se calcula el ángulo ψ

$$\psi = -atan2\left(M_{med\_rot\_y}, M_{med\_rot\_x}\right) + \psi_{dec} \tag{2.97}$$

Una vez que se obtienen los 3 ángulos de euler, se recalcula $\boldsymbol{q}$ para tener en cuenta el ángulo ψ.
En cuanto a la posición, la altura se establece a partir del barómetro y para los ejes norte y este se utiliza la lectura del GPS. Todos los errores de offset ($\Delta\boldsymbol{Ang}_{bias}, \Delta\boldsymbol{Vel}_{bias}$ y $\boldsymbol{M}_{bias}$), se inicializan a 0. Esto es así porque los valores de offset que se obtienen en la calibración ya se compensan en otro módulo.

2.4.5.2 Matriz de covarianzas

La matriz de covarianzas se inicializa a partir de parámetros definidos por el usuario.
Para la velocidad horizontal:

$$P_{5,5} = P_{6,6} = EKF2\_GPS\_V\_NOISE^2 \tag{2.98}$$

Para la velocidad vertical:

$$P_{7,7} = (1.5 \cdot EKF2\_GPS\_V\_NOISE)^2 \tag{2.99}$$

Para la posición horizontal:

$$P_{8,8} = P_{9,9} = EKF2\_GPS\_P\_NOISE^2 \tag{2.100}$$

[20] La declinación magnética puede conocerse a partir de la posiciín que ofrece el GPS.

Para la vertical:

$$P_{10,10} = EKF2\_BARO\_NOISE^2 \tag{2.101}$$

Para el error de offset del giróscopo

$$P_{11,11} = P_{12,12} = P_{13,13} = (EKF2\_GBIAS\_INIT \cdot \Delta t)^2 \tag{2.102}$$

Para el error de offset del acelerómetro:

$$P_{14,14} = P_{15,15} = P_{16,16} = (EKF2\_ABIAS\_INIT \cdot \Delta t)^2 \tag{2.103}$$

Tanto para los estados $\boldsymbol{M}_{NED}$ como los de $\boldsymbol{M}_{bias}$:

$$P_{17,17} = P_{18,18} = P_{19,19} = P_{20,20} = P_{21,21} = P_{22,22} = (EKF2\_MAG\_NOISE)^2 \tag{2.104}$$

Para la orientación se conoce el error en el vector de orientación (el que señala la dirección de rotación) y por tanto este tendrá que ser convertido a cuaterniones:

$$rot\_vec_{var} = \begin{bmatrix} EKF2\_ANGERR\_INIT^2 & 0 & 0 \\ 0 & EKF2\_ANGERR\_INIT^2 & 0 \\ 0 & 0 & EKF2\_ANGERR\_INIT^2 \end{bmatrix} \tag{2.105}$$

La transformación del vector de rotación a cuaterniones es la siguiente:

$$\boldsymbol{q}(\boldsymbol{rot\_vec}) = \begin{bmatrix} cos\left(\frac{\|\boldsymbol{rot\_vec}\|}{2}\right) \\ \frac{\boldsymbol{rot\_vec}}{\|\boldsymbol{rot\_vec}\|} sin\left(\frac{\|\boldsymbol{rot\_vec}\|}{2}\right) \end{bmatrix} \tag{2.106}$$

A esta función se le realiza el jacobiano:

$$G = \frac{\partial \boldsymbol{q}(\boldsymbol{rot\_vec})}{\partial \boldsymbol{rot\_vec}} = \begin{bmatrix} \frac{\partial q_0}{\partial rot\_vec_x} & \cdots & \frac{\partial q_0}{\partial rot\_vec_z} \\ \vdots & \ddots & \vdots \\ \frac{\partial q_3}{\partial rot\_vec_x} & \cdots & \frac{\partial q_3}{\partial rot\_vec_z} \end{bmatrix} \tag{2.107}$$

Finalmente se transfoma la matriz de covarianzas del vector de rotación a la matriz de covarianzas de los cuaterniones:

$$q_{var} = G\ rot\_vec_{var}\ G^T \tag{2.108}$$

Para terminar de inicializar la matriz de covarianzas, se le asigna este cálculo obtenido:

$$P_{1:4,1:4} = q_{var} \tag{2.109}$$

3 Modelado del tiltrotor

En este capítulo se va a obtener el modelo matemático de un tilrotor de dos rotores, el cual se utilizará en el capítulo posterior para el diseño analítico de los controladores. Además, ese modelo será implementado en *Simulink*, para más tarde, simularlo con los controladores.

3.1 Modelo matemático del tiltrotor

El tiltrotor que se va a modelar (ver figura 3.1) está formado por dos rotores separados a la misma distancia (l_y) del centro de masas y que giran a una velocidad ω_1 y ω_2. Estos y el motor que los hace girar pueden rotar (ángulos α_1 y α_2) sobre el mismo eje situado a una distancia l_z del centro de masas. Se han tomado las siguientes simplificaciones:

- No se contemplarán las fuerzas aerodinámicas como la resistencia aerodinámica. Aproximación mayor cuanto mayor sea la velocidad a la que se mueve el vehículo.
- En una implementación real no existiría un único sólido rígido, sino 5 que se mueven entre sí (el fuselaje, dos hélices y los dos motores que hacen girarlas). Si se contemplasen, el modelo matemático sería más complicado. En su lugar se tendrá un solo sólido rígido en el que actúan fuerzas que varían temporalmente en orientación y magnitud.

Los parámetros dinámicos geométricos del vehículo son los siguientes:

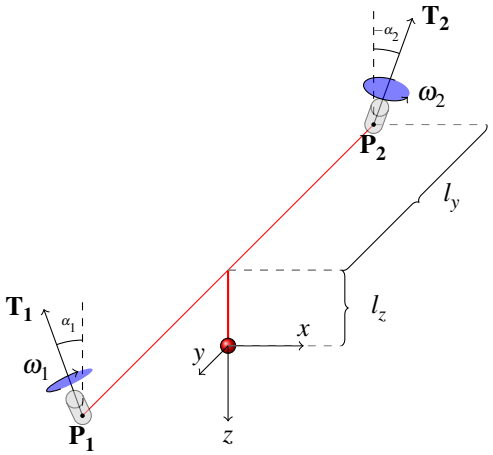

Figura 3.1 Esquema del vehículo. La esfera roja indica su centro de masas.

Símbolo	Valor	Descripción
τ_m	$0.01s$	Constante de tiempo de los motores que hacen girar las hélices
τ_α	$0.01s$	Constante de tiempo de los servomotores de inclinación
J_x	$0.03kg \cdot m^2$	Inercia en el eje x
J_y	$0.05kg \cdot m^2$	Inercia en el eje y
J_z	$0.09kg \cdot m^2$	Inercia en el eje z
m	1Kg	Masa total del vehículo
l_y	0.5 m	Distancia en el eje y desde el C.M. hasta los rotores
l_z	0.25 m	Distancia en el eje z desde el C.M. hasta los rotores
ω_{max}	1100 rad/s	Velocidad máxima de los motores
c_m	0	Coeficiente de arrastre
c_T	$8.0992 \cdot 10^{-6} \frac{N \cdot m}{(rad/s)^2}$	Coeficiente de empuje

Tabla 3.1 Parámetros del tiltrotor.

Se considera un solo sólido rígido en el que actúan las siguientes fuerzas y pares:

- $\boldsymbol{F}_g$. Fuerza gravitatoria. En ejes inerciales tiene la siguiente expresión:

$$\boldsymbol{F}_g = \begin{bmatrix} 0 \\ 0 \\ g\,m \end{bmatrix} \tag{3.1}$$

- $\boldsymbol{T}_1$, $\boldsymbol{T}_2$. Empuje de cada uno de los rotores. Son fuerzas aplicadas en $\boldsymbol{P}_1$ y $\boldsymbol{P}_2$. Se expresan en ejes cuerpo de la siguiente manera:

$$\boldsymbol{T}_i = \begin{bmatrix} -\sin(\alpha_i)T_i \\ 0 \\ -\cos(\alpha_i)T_i \end{bmatrix} \tag{3.2}$$

$$T_i = c_T\ \omega_i^2 \tag{3.3}$$

$$\boldsymbol{P}_1 = \begin{bmatrix} 0 \\ l_y \\ -l_z \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{P}_2 = \begin{bmatrix} 0 \\ -l_y \\ -l_z \end{bmatrix} \tag{3.5}$$

- $\boldsymbol{\tau}_{d1}$, $\boldsymbol{\tau}_{d2}$. Par de reacción de cada uno de los rotores. El rotor 1 gira en sentido positivo del eje z (cuando $\alpha_1 = 0$). Por reacción se genera un par en el sentido contrario que es función de la velocidad de rotación al cuadrado y de una constante c_m. El rotor 2 gira en sentido contrario para poder compensar el par del otro.

$$\boldsymbol{\tau}_{d1} = \begin{bmatrix} -\sin(\alpha_1)\tau_{d1} \\ 0 \\ -\cos(\alpha_1)\tau_{d1} \end{bmatrix} \tag{3.6}$$

$$\boldsymbol{\tau}_{d2} = \begin{bmatrix} \sin(\alpha_2)\tau_{d2} \\ 0 \\ \cos(\alpha_2)\tau_{d2} \end{bmatrix} \tag{3.7}$$

$$\tau_{di} = c_m\ \omega_i^2 \tag{3.8}$$

Las fuerzas que están expresadas en ejes cuerpo, deben de ser convertidas a ejes inerciales para poder ser sumadas en la ecuación de traslación. Esto se realiza mediante una matriz de rotación que va a ser función de

los ángulos de euler:

$$R^{B\to I} = \begin{bmatrix} cos(\theta)cos(\psi) & sin(\phi)sin(\theta)cos(\psi) - cos(\phi)sin(\psi) & cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) \\ cos(\theta)sin(\psi) & sin(\phi)sin(\theta)sin(\psi) + cos(\phi)cos(\psi) & cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi) \\ -sin(\theta) & sin(\phi)cos(\theta) & cos(\phi)cos(\theta) \end{bmatrix} \tag{3.9}$$

Ecuación de traslación en el sistema inercial:

$$m\,\ddot{\mathbf{P}} = \Sigma \boldsymbol{F} \tag{3.10}$$

$$m\,\ddot{\mathbf{P}} = \boldsymbol{F}_g + R\,(\boldsymbol{T}_1 + \boldsymbol{T}_2) \tag{3.11}$$

La ecuación de rotación se expresará en coordenadas cuerpo, ya que el tensor de inercias expresado en estos ejes es constante:

$$\Sigma \boldsymbol{\tau} = I\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (I\,\boldsymbol{\omega}) \tag{3.12}$$

Donde el sumatorio de pares es igual a:

$$\Sigma \boldsymbol{\tau} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \boldsymbol{\tau}_{d1} + \boldsymbol{\tau}_{d1} + \boldsymbol{P}_1 \times \boldsymbol{T}_1 + \boldsymbol{P}_2 \times \boldsymbol{T}_2 \tag{3.13}$$

Siendo I el tensor de inercia, que en este caso es diagonal:

$$I = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} \tag{3.14}$$

Se despeja la aceleración angular y se desarrolla:

$$\dot{\boldsymbol{\omega}} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1}\Sigma\boldsymbol{\tau} + I^{-1}\boldsymbol{\omega} \times (I\boldsymbol{\omega}) = \begin{bmatrix} \frac{1}{J_x}\tau_x \\ \frac{1}{J_y}\tau_y \\ \frac{1}{J_z}\tau_z \end{bmatrix} + \begin{bmatrix} \frac{J_y - J_z}{J_x} qr \\ \frac{J_z - J_x}{J_y} pr \\ \frac{J_x - J_y}{J_z} pq \end{bmatrix} \tag{3.15}$$

Si supone pequeñas velocidades se puede simplificar:

$$\dot{\boldsymbol{\omega}} = \begin{bmatrix} \frac{1}{J_x}\tau_x \\ \frac{1}{J_y}\tau_y \\ \frac{1}{J_z}\tau_z \end{bmatrix} \tag{3.16}$$

3.2 Modelo del tiltrotor en *Matlab/Simulink®*

En la figura 3.2 se muestra los bloques utilizados en *Simulink* para simular el tiltrotor. Las entradas del sistema son dos perturbaciones, una en par (*tau_pert*) y otra en fuerza (*F_pert*) y las 4 señales de actuación (las inclinaciones y las velocidades de los rotores). Estas últimas generarán pares y fuerzas de acuerdo con las ecuaciones 3.4, 3.5, 3.6 y 3.7 en el bloque *Generación de Fuerzas y pares* que está implementado en *Matlab* en el archivo *genera_tau_f.m*.

Código 3.2.1: Generador de fuerzas y pares (*genera_tau_f.m*)

```
1  function out = genera_tau_f(in)
2
3      omega1=in(1);
4      omega2=in(2);
5      alfa1=in(3);
6      alfa2=in(4);
7
8      global param
```

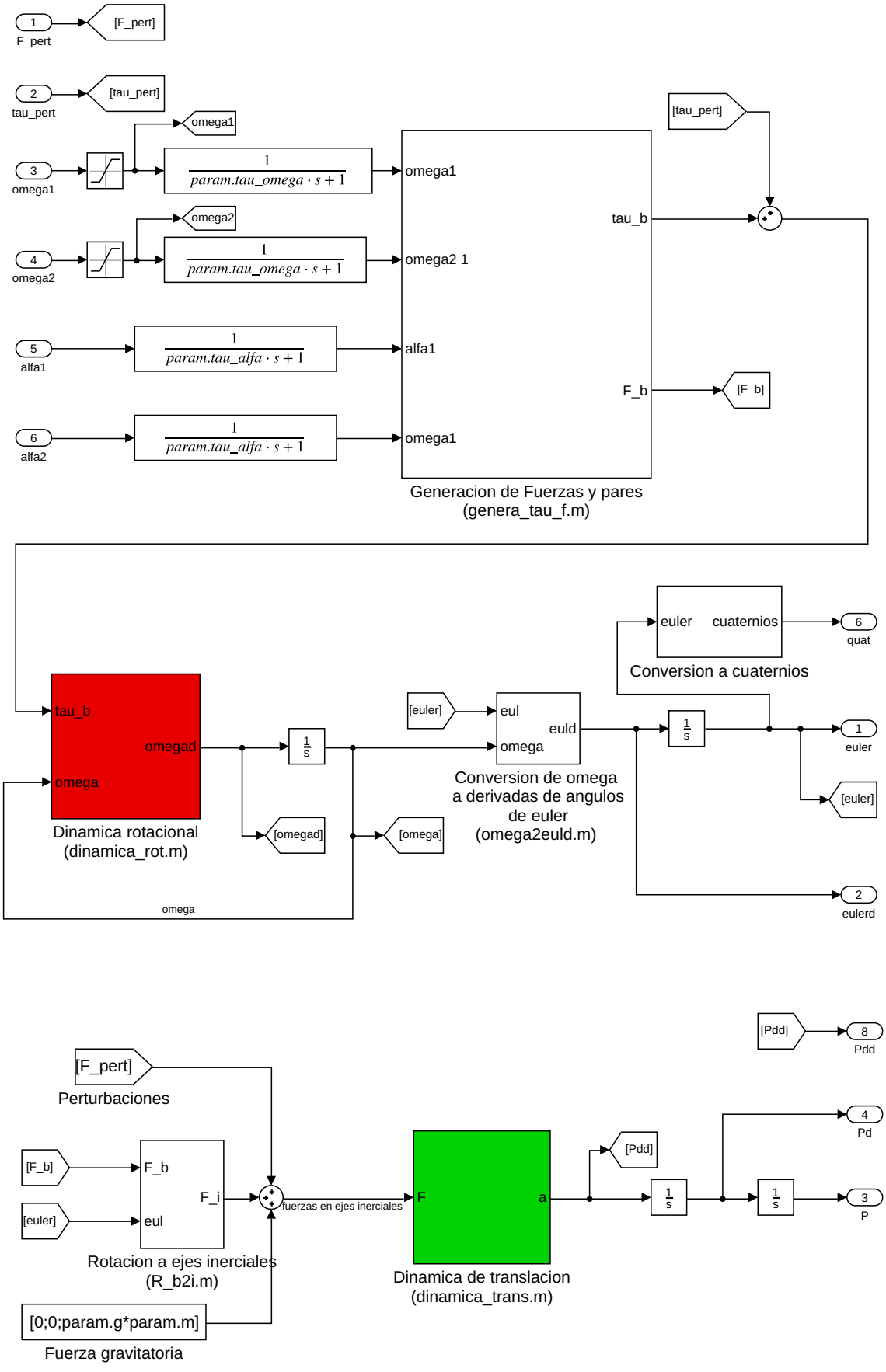

Figura 3.2 Subsistema de dinámica del tiltrotor.

```
    ly=param.ly;
    lz=param.lz;

    t1=omega1*omega1*param.cT;
    t2=omega2*omega2*param.cT;
    T1=[-t1*sin(alfa1);   0; -t1*cos(alfa1)];
    T2=[-t2*sin(alfa2);   0; -t2*cos(alfa2)];
    taud1=omega1*omega1*param.cm;
    taud2=omega2*omega2*param.cm;

tau=[...
    ly*t2*cos(alfa2)-ly*t1*cos(alfa1)-taud1*sin(alfa1)+taud2*sin(alfa2);
    lz*t1*sin(alfa1)+lz*t2*sin(alfa2)                                  ;
    ly*t1*sin(alfa1)-ly*t2*sin(alfa2)-taud1*cos(alfa1)+taud2*cos(alfa2)];

out=[tau;T1+T2];
     if alfa1~=0

     end
end
```

Los pares generados por las señales de actuación y los de perturbación, ambos en coordenadas cuerpo, se suman y entran en el bloque *Dinámica rotacional*. Este genera las las velocidades angulares según la ecuación 3.15 y está implementado en el archivo *dinamica_rot.m*:

Código 3.2.2: Dinámica de rotación (*dinamica_rot.m*)

```
function out=dinamica_rot(in)
    tau_b=[in(1); in(2); in(3)];
    omega=[in(4); in(5); in(6)];
    p=omega(1);
    q=omega(2);
    r=omega(3);

    global param
    Jx=param.Jx;
    Jy=param.Jy;
    Jz=param.Jz;
    omegad= tau_b./[param.Jx; param.Jy; param.Jz] + [(Jy-Jz)/Jx*q*r;
    (Jz-Jx)/Jy*p*r;   (Jx-Jy)/Jz*p*q];

    out=omegad;

end
```

La salida de este bloque se integra obteniéndose las velocidades angulares que serán realimentadas a la entrada. Estas velocidades angulares serán convertidas a derivadas de ángulos de euler multiplicándolas por una matriz que depende de los ángulos de euler (la obtención de está matriz se muestra en la ecuación 7 de [7]). Dicha multiplicación se realiza en el siguiente archivo:

Código 3.2.3: Conversión de velocidades angulares en ejes cuerpo a derivadas de ángulo de euler(*omega2euld.m*)

```
function out = omega2euld(in)

    roll=in(1);
    pitch=in(2);
    yaw=in(3);
    p=in(4);
    q=in(5);
    r=in(6);
    A=[1,    sin(roll)*tan(pitch),   cos(roll)*tan(pitch);
       0,    cos(roll),              -sin(roll);
       0,    sin(roll)/cos(pitch),   cos(roll)/cos(pitch)];
    out=A*[p;q;r];
end
```

Finalmente se integra la derivada temporal de los ángulos de euler y se obtiene la orientación expresada en ángulos de euler.

Para la dinámica translacional es necesario obtener todas las fuerzas en ejes inerciales y para ello, las fuerzas en ejes cuerpo generadas por los actuadores se rotan multiplicándolas por la matriz de rotación 3.9:

Código 3.2.4: Rotación de ejes cuerpo a inerciales (*R_b2i.m*)

```
function out= R_b2i(in)
 F=[in(1); in(2); in(3)];
 eul=[in(4); in(5); in(6)];


 roll=eul(1);
 pitch=eul(2);
 yaw=eul(3);

 R=[cos(pitch)*cos(yaw), sin(roll)*sin(pitch)*cos(yaw)-cos(roll)*sin(yaw),
  ↪   cos(roll)*sin(pitch)*cos(yaw)+sin(roll)*sin(yaw);
    cos(pitch)*sin(yaw), sin(roll)*sin(pitch)*sin(yaw)+cos(roll)*cos(yaw),
  ↪   cos(roll)*sin(pitch)*sin(yaw)-sin(roll)*cos(yaw);
    -sin(pitch)          ,sin(roll)*cos(pitch)
  ↪   ,cos(roll)*cos(pitch)                          ];

 out=R*F;
end
```

A estas últimas se les suma tanto la fuerza de perturbación como la gravedad y entran en el bloque *Dinámica de translacion* que implementa la ecuación 3.11:

Código 3.2.5: Dinámica de translación (*dinamica_trans.m*)

```
function out = dinamica_trans(in)

   F=[in(1);in(2); in(3)];
   global param
   a=F./param.m;
   out=a;

end
```

La salida de este bloque es la aceleración lineal, que se integra dos veces obteniéndose la posición.

4 Diseño de controladores

En este este capítulo se busca controlar el modelo de tiltrotor descrito en el capítulo anterior, con unos controladores que sean lo más parecidos posible a los que hay en *PX4*. Se verá que el único cambio necesario está en el control *mixer*. Dicho controlador, junto con los demás, serán implementados en *Simulink* para comprobar si se cumplen las especificaciones establecidas en el posterior diseño de controladores, para el cual, se empelaran dos métodos: uno analítico y otro iterativo.

4.1 Creación de un control *mixer* para el tiltrotor

El objetivo de esta sección es hallar una función que calcule las señales de actuación necesarias para que se produzcan unas fuerzas y pares dados en el sistema de referencia del vehículo. Para empezar, se halla la relación inversa. Por un lado, el par total ($\Sigma\boldsymbol{\tau}$) es la suma de los pares que producen los dos rotores en el eje de giro ($\boldsymbol{\tau}_d$) y los pares que producen los empujes ($\boldsymbol{T}_1$ y $\boldsymbol{T}_2$) al estar aplicados alejados del centro de masas ($\boldsymbol{\tau}_F$).

$$\Sigma\boldsymbol{\tau} = \boldsymbol{\tau}_F + \boldsymbol{\tau}_d \tag{4.1}$$

$$\boldsymbol{\tau}_F = \boldsymbol{\tau}_{F1} + \boldsymbol{\tau}_{F2} = \boldsymbol{P}_1 \times \boldsymbol{T}_1 + \boldsymbol{P}_2 \times \boldsymbol{T}_2 \tag{4.2}$$

$$\boldsymbol{\tau}_d = \boldsymbol{\tau}_{d1} + \boldsymbol{\tau}_{d2} \tag{4.3}$$

Por otro lado, la fuerza total ($\boldsymbol{F}$) es la suma de esos dos empujes:

$$\boldsymbol{F} = \boldsymbol{T}_1 + \boldsymbol{T}_2 \tag{4.4}$$

Se desarrolla y se obtiene:

$$\Sigma\boldsymbol{\tau} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \boldsymbol{\tau}_F + \boldsymbol{\tau}_d = \begin{bmatrix} l_y\, T_2\, cos(\alpha_2) - l_y\, T_1\, cos(\alpha_1) - \tau_{d1}\, sin(\alpha_1) + \tau_{d2}\, sin(\alpha_2) \\ l_z\, T_1\, sin(\alpha_1) + lz\, T_2\, sin(\alpha_2) \\ l_y\, T_1\, sin(\alpha_1) - l_y\, T_2\, sin(\alpha_2) - \tau_{d1}\, cos(\alpha_1) + \tau_{d2}\, cos(\alpha_2) \end{bmatrix} \tag{4.5}$$

$$\mathbf{F} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} -T_1\, sin(\alpha_1) - T_2\, sin(\alpha_2) \\ 0 \\ -T_1\, cos(\alpha_1) - T_2\, cos(\alpha_2) \end{bmatrix} \tag{4.6}$$

El sistema de ecuaciones que ha resultado es no lineal. Por tanto se harán algunas simplificaciones para poder resolverlo. Para valores pequeños de α_1 y α_2 se pueden tomar las siguientes aproximaciones:

$$sin(\alpha_i) \approx \alpha_i \tag{4.7}$$

$$cos(\alpha_i) \approx 1 \tag{4.8}$$

Además se supondrá que los términos de los pares de reacción τ_{d1} y τ_{d2} son mucho menores que los otros a los que está sumando y por tanto se igualan a 0.

Hay que notar que existen solo 4 señales de actuación $\big(T_1(\omega_1), T_2(\omega_2), \alpha_1$ y $\alpha_2\big)$, mientras que el vehículo posee 6 grados de libertad (3 rotaciones y 3 traslaciones). Por tanto este es un sistema subactuado, y no es posible controlar todos los grados de libertad al mismo tiempo. Dicho esto, no se tendrán en cuenta las

componentes x e y del vector de fuerzas deseado (ecuación 4.6) y solo afectará la componente F_z para el cálculo de las señales de actuación. Dicha elección se ha hecho en base a que F_y siempre es nulo y F_x es menor que F_z con valores pequeños de α_1 y α_2.

Aplicando dichas suposiciones, las ecuaciones quedan de la siguiente manera:

$$\tau_x = l_y\, T_2\ - l_y\, T_1 \tag{4.9a}$$
$$\tau_y = l_z\, T_1\, \alpha_1 + l_z\, T_2\, \alpha_2 \tag{4.9b}$$
$$\tau_z = l_y\, T_1\, \alpha_1 - l_y\, T_2\, \alpha_2 \tag{4.9c}$$
$$F_z = -T_1 - T_2 \tag{4.9d}$$

A partir de las equaciones 4.9a y 4.9d se hallan los dos empujes:

$$\tau_x = l_y\, T_2\ - l_y\, (-F_z - T_2)\ \rightarrow \tag{4.10}$$

$$T_2 = \frac{\tau_x + l_y\, F_z}{2 l_y} = \frac{1}{2 l_y}\tau_x - \frac{1}{2}F_z \tag{4.11}$$

$$T_1 = F_z - T_2 = -\frac{1}{2 l_y}\tau_x - \frac{1}{2}F_z \tag{4.12}$$

De la ecuación 4.9c se despeja el siguiente término,

$$T_2\, \alpha_2 = \frac{l_y\, T_1\, \alpha_1 - \tau_z}{l_y} \tag{4.13}$$

y se sustituye en la ecuación 4.9b:

$$\tau_y = l_z\, T_1\, \alpha_1 + lz\, \frac{l_y\, T_1\, \alpha_1 - \tau_z}{l_y} \tag{4.14}$$

Despejando α_1 queda:

$$\alpha_1 = \frac{\tau_y + \frac{lz\, \tau_z}{l_y}}{2\, l_z\, T_1} \tag{4.15}$$

Por último, se reordena esta última ecuación para dejar clara la dependencia con los pares deseados τ_y y τ_z:

$$\alpha_1 = \frac{1}{2\, l_z\, T_1}\tau_y + \frac{1}{2\, l_y\, T_1}\tau_z \tag{4.16}$$

De forma similar con α_2:

$$\alpha_2 = \frac{1}{2\, l_z\, T_1}\tau_y - \frac{1}{2\, l_y\, T_1}\tau_z \tag{4.17}$$

4.2 Simulación de los controladores en Matlab/Simulink®

En esta sección se va a incluir el modelo de *Simulink* utilizado y las funciones en *Matlab* que modelan los controladores de *PX4*. Una vista general del modelo puede verse en la figura 4.1. Se puede comprobar que este esquema es el mismo al que hay implementado en *PX4* que se ilustró en la figura 2.6.

Antes de empezar ha describir las partes, hay que aclarar que a lo largo del código se hará uso de una variable global llamada *param*. Esta es una estructura que en sus campos almacena los valores de todos los

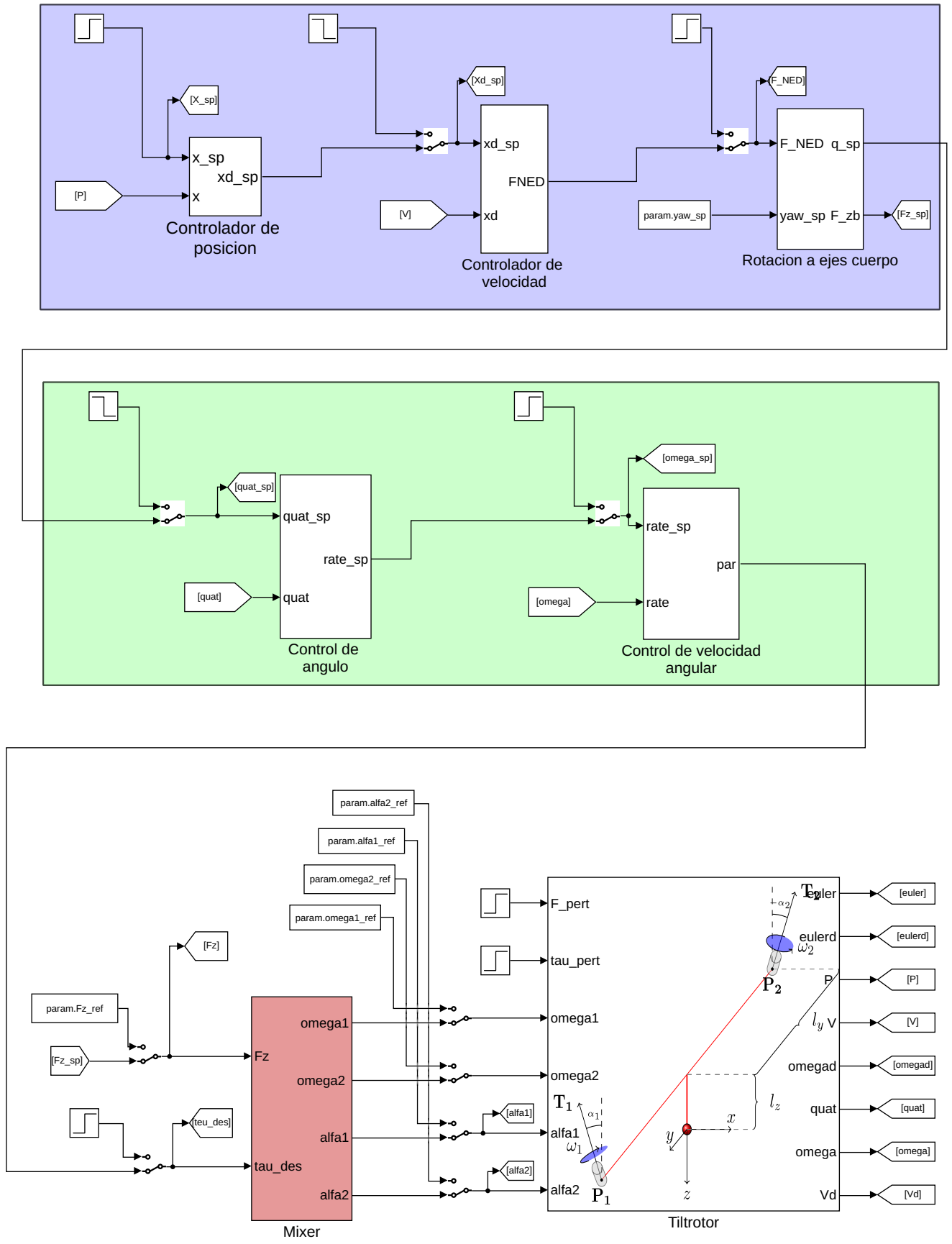

Figura 4.1 Esquema de simulink.

parámetros usados en la simulación. Algunos de estos son las ganancias de los controladores, las opciones del experimento como la amplitud de las referencias y los parámetros dinámicos del tiltrotor. Algunos de ellos están en mayúsculas, esto es porque corresponden diretamente con parámetros de *PX4*[1]. A dicha estructura global, se le da valor mediante el script *parametros.m* que se se encuentra en el apéndice A, de manera que los experimentos aquí mostrados sean totalmente replicables con la información que hay en este documento.

A la entrada de cada uno de los controladores existen unos conmutadores, para poder cambiar entre una referencia fija, o una que proviene del controlador que esté a un nivel superior. Todos los controladores están implementados mediante funciones estáticas escritas en *Matlab* (utilizando el bloque *Interpreted Matlab Function*), de manera que no tengan variables *persistent*. Para el cálculo de la integral del error o la estimación de las derivadas, se utilizan bloques dinámicos como integradores o funciones de transferencia discretas.

Para empezar, el control *mixer* que se obtuvo en la sección anterior se muestra a continuación escrito en *Matlab*:

Código 4.2.1: Control mixer (*mixer_3d.m*)

```
function out= mixer_3d(in)
    taux=in(1);
    tauy=in(2);
    tauz=in(3);
    Fz=in(4);
    global param

    ly=param.ly;
    lz=param.lz;

    T2= taux/(2*ly) + Fz/2;
    T1=-taux/(2*ly) + Fz/2;
    if T1<0
         T1=0;
    end
    if T2<0
         T2=0;
    end

    if T1<0.01
        alfa1=0;
    else
        alfa1=1/(2*lz*T1)*tauy  + 1/(2*ly*T1)*tauz;
        alfa1=min(max(alfa1,-param.max_alfa),param.max_alfa);
    end



    if T2<0.01
        alfa2=0;
    else
        alfa2=1/(2*lz*T2)*tauy  - 1/(2*ly*T2)*tauz;
        alfa2=min(max(alfa2,-param.max_alfa),param.max_alfa);
    end

    omega1=sqrt(T1/param.cT);
    omega2=sqrt(T2/param.cT);

    out=[omega1;omega2;alfa1;alfa2];
    if taux>0.1
    end
end
```

En cuanto al controlador de velocidad angular es un PID, y se puede ver en el siguiente código que el término integral y el derivativo se reciben como entradas:

[1] Todos los parámetros están descritos en este enlace: dev.px4.io/v1.9.0/en/advanced/parameter_reference.html

Código 4.2.2: Controlador de velocidad angular (*mc_att_rate_control.m*)

```
function out=mc_att_rate_control(in)
    rate_sp=[in(1);in(2); in(3)]; % Velocidad angular deseada
    rate=[in(4);in(5); in(6)]; % Velocidad angular leida
    e_int=[in(7);in(8); in(9)]; % Integral del error
    rate_d=[in(10);in(11); in(12)]; % Estimación de la aceleración angular

    global param
    kp=[param.MC_ROLLRATE_P;param.MC_PITCHRATE_P;param.MC_YAWRATE_P];
    ki=[param.MC_ROLLRATE_I;param.MC_PITCHRATE_I;param.MC_YAWRATE_I];
    kd=[param.MC_ROLLRATE_D;param.MC_PITCHRATE_D;param.MC_YAWRATE_D];

    tau= kp.*( rate_sp_y - rate_y) + kd.*( 0 - rate_d) + ki.*e_int;

    % Saturaciones
    tau(1)=min(  param.max_taux, tau(1));
    tau(1)=max(-param.max_taux, tau(1));
    tau(2)=min(  param.max_tauy, tau(2));
    tau(2)=max(-param.max_tauy, tau(2));
    tau(3)=min(  param.max_tauz, tau(3));
    tau(3)=max(-param.max_tauz, tau(3));

    out=tau;
end


```

La integral y la derivada se obtienen mediante bloques de *Simulink* y se muestran en la figura 4.2. Se puede comprobar que la estimación de la aceleración angular se ha implementado de la misma forma que en *PX4*, mediante un filtro discreto de paso de bajas y una derivada discreta (ver subsección 2.3.5). Además, a la medida de la velocidad angular se le ha añadido un retraso igual a la mitad del tiempo de muestreo [2]. Por último, existe un integrador que tiene como entrada el error y está saturado con los mismos parámetros que en *PX4*: *MC_RR_INT_LIM*, *MC_PR_INT_LIM* y *MC_YR_INT_LIM* para los ejes x, y y z respectivamente.

En el siguiente código se muestra la implementación del controlador de orientación descrito en la sección 2.3.4.

Código 4.2.3: Controlador de ángulo (*mc_att_control.m*)

```
function out= mc_att_control(in)
    q_sp= [in(1);in(2);in(3);in(4)]'; % Orientación de referencia
    q=    [in(5);in(6);in(7);in(8)]'; % Orientación actual

    % Simulink inicializa los cuaternios a 0, algo que puede dar problemas
    if norm(q)==0
        q=q_sp;
    end

    global param

    Kp=[param.MC_ROLL_P;param.MC_PITCH_P;(param.MC_ROLL_P+param.MC_PITCH_P)/2];

    % Verison simplificada de la ley de control angular de PX4. Cuando solo
    % se controla un eje no deberia de haber diferencias.
    qe=quatmultiply(quatinv(q),q_sp)';
    omega_sp=2*Kp.*sign(qe(1)).*qe(2:4);

    limites=[param.MC_ROLLRATE_MAX;param.MC_PITCHRATE_MAX;param.MC_YAWRATE_MAX]*pi/180;
    omega_sp=min(omega_sp, limites);
    omega_sp=max(omega_sp,-limites);

    out=omega_sp;
end
```

[2] Este valor se ha escogido al comprobar que produce unos resultados parecidos con la simulación de *PX4/Gazebo* que se verá en el capítulo siguiente

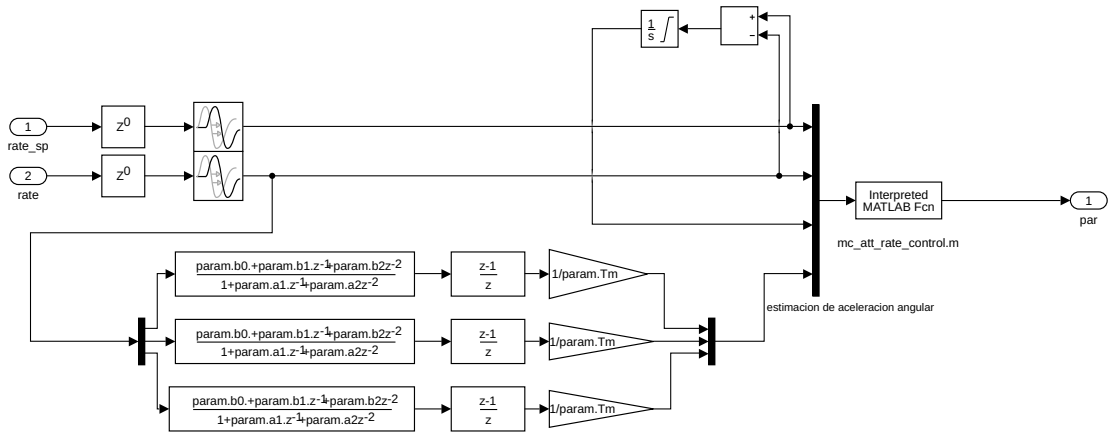

Figura 4.2 Subsistema de control de velocidad angular.

En la figura 4.3 se puede ver que a la lectura de la orientación se le ha añadido un retraso de 6 ms para añadir el efecto del estimador de estados de *PX4*.

La siguiente función es el controlador de velocidad:

Código 4.2.4: Controlador de velocidad (*vel_control.m*)

```
function out=vel_control(in)
    Vel=[in(1);in(2);in(3)];
    Vel_sp=[in(4);in(5);in(6)];
    Acc=[in(7);in(8);in(9)];
    e_int=[in(10);in(11);in(12)];

    global param

    Kp=param.MPC_Z_VEL_P;
    Ki=param.MPC_Z_VEL_I;
    Kd=param.MPC_Z_VEL_D;

    %%% Controlador en el eje z %%%
    Fz=Kp*(Vel_sp(3)-Vel(3)) + Kd*(0 - Acc(3)) + Ki*e_int(3) - param.MPC_THR_HOVER;

    % Antiwindup
    if (abs(e_int(3)*Ki) > abs(param.MPC_THR_MAX) )    || ...
             ( Fz > 0 && e_int(3)<0 )  || ...
             ( Fz < -param.MPC_THR_MAX && e_int(3)>0 )
        integrar(3)=0;
    else
        integrar(3)=Vel_sp(3)-Vel(3);
    end

    Fz=max(Fz,-param.MPC_THR_MAX);
```

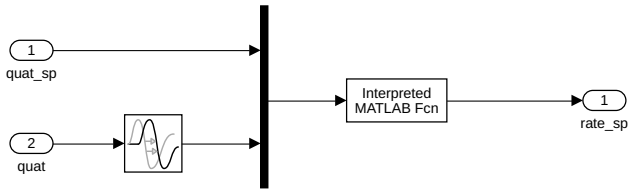

Figura 4.3 Subsistema de control de ángulo.

```
26      Fz=min(Fz,0);
27
28      %%% Controlador en los ejes XY %%%
29      Kp=param.MPC_XY_VEL_P;
30      Ki=param.MPC_XY_VEL_I;
31      Kd=param.MPC_XY_VEL_D;
32      Fxy=Kp*(Vel_sp(1:2)-Vel(1:2)) + Kd*(0 - Acc(1:2)) + Ki*e_int(1:2);
33
34      % Saturaciones
35      Fxy_mod=norm(Fxy);
36      saturacion1=sqrt(param.MPC_THR_MAX^2 - Fz^2);
37      saturacion2=abs(Fz*tand(param.MPC_TILTMAX_AIR));
38      saturacion_min=min(saturacion1,saturacion2);
39      Fxy_sat=Fxy;
40      if Fxy_mod>saturacion_min
41          Fxy_sat=Fxy/Fxy_mod*saturacion_min;
42      end
43
44      % La variable 'integrar' será una salida que se integrará fuera de este
45      % bloque y entrará como 'e_int'
46      if param.MPC_XY_VEL_I>0
47           integrar(1:2)=Vel_sp(1:2)-Vel(1:2)  -2/Kp*(Fxy-Fxy_sat);
48      else
49          integrar(1:2)=[0,0];
50      end
51
52      out=[Fxy_sat;Fz;integrar'];
53
54  end
```

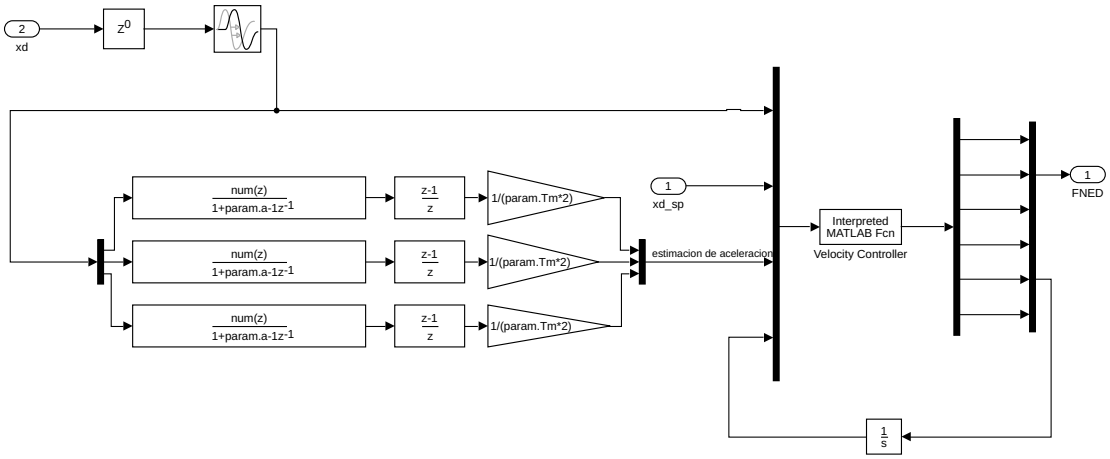

Figura 4.4 Subsistema de control de velocidad.

A la medida de la velocidad se le ha añadido un retraso de 16 ms y se le ha estimado la derivada mediante un filtro discreto de un solo polo (ver figura 4.4).

Por último, el controlador de posición:

Código 4.2.5: Controlador de posición (*pos_control.m*)

```
function out = pos_control(in)

    x_sp=[in(1);in(2);in(3)];
    x=[in(4);in(5);in(6)];

    global param

    xd_sp=[param.MPC_XY_P;  param.MPC_XY_P; param.MPC_Z_P].*(x_sp-x);

    % Saturación
    VELxy=[xd_sp(1); xd_sp(2)];
    velxy=norm(VELxy);
    if velxy > param.MPC_XY_VEL_MAX
        VELxy=VELxy/velxy*param.MPC_XY_VEL_MAX;
    end

    out=[VELxy; xd_sp(3)];
end
```

4.3 Diseño analítico de controladores

Una vez obtenido el modelo dinámico del tiltrotor y hallado un método para aplicar fuerzas y pares deseados en ejes cuerpo, se diseñará los controladores de forma analítica. Se empezará diseñando y evaluando desde los

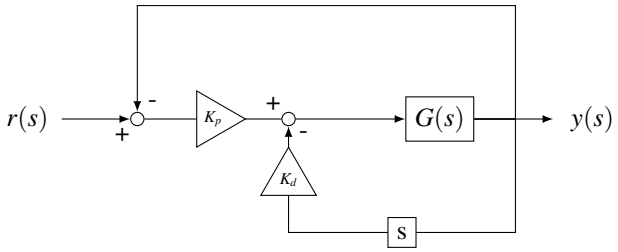

Figura 4.5 Diagrama de bloques del controlador. La acción derivativa se calcula solo sobre la derivada de la salida.

controladores más internos a los más externos. Siempre que se pase a un controlador superior, este se diseñará con un ancho de banda menor y haciendo simplificaciones del sistema subordinado que ve el controlador.

4.3.1 Control de velocidad angular

Se va a tomar como modelo dinámico la ecuación de rotación simplificada (ecuación 3.16). Esta se puede expresar en forma de función de transferencia:

$$G(s) = \frac{\Omega(s)}{\tau(s)} = \frac{1}{J\,s} \tag{4.18}$$

Si se aplica al eje y y se añade la dinámica de los servomotores queda:

$$G_y(s) = \frac{q(s)}{\tau_y(s)} = \frac{1}{J_y\,s(\tau_\alpha s+1)} \tag{4.19}$$

El controlador que se diseñará será el mostrado en la figura 4.5, que es una simplificación del descrito en la subsección 2.3.5, del que se ha eliminado el término integral y el de feed-forward para facilitar el diseño. Mediante álgebra de bloques, la función de transferencia de bucle cerrado en el seguimiento queda:

$$G_{BC}(s) = \frac{K_p G(s)}{1+K_p G(s)+K_d G(s)s} = \frac{K_p}{J_x\tau_m s^2+(J_x+K_d)s+K_p} = \frac{\frac{K_p}{J_x\tau_m}}{s^2+\frac{(J_x+K_d)}{J_x\tau_m}s+\frac{K_p}{J_x\tau_m}} \tag{4.20}$$

Se compara con un sistema de segundo orden donde ξ es el factor de amortiguamiento y ω_n es la frecuencia de oscilación natural:

$$G(s) = \frac{\omega_n^2}{\omega_n^2+2\omega_n\xi+s^2} \tag{4.21}$$

$$\omega_n = \sqrt{\frac{K_p}{J_x\tau_m}} \tag{4.22}$$

$$\xi = \frac{(J_x+K_d)}{2\omega_n J_x\tau_m} \tag{4.23}$$

Se despejan las dos ganancias del controlador:

$$K_p = \omega_n^2 J_x\tau_m \tag{4.24}$$

$$K_d = \xi 2\omega_n J_x\tau_m - J_x \tag{4.25}$$

Para que resulte un sistema críticamente amortiguado (dos polos reales e iguales) se establece $\xi = 1$. Además se pedirá un tiempo de subida de 0.05s. Este tiempo de subida se ha elegido en base a resultados experimentales, ya que si se pide una respuesta demasiado rápida, las señales de actuación serían muy altas y las aproximaciones tomadas en la creación del control *mixer* (ecuación 4.8), tendrían un efecto más negativo.

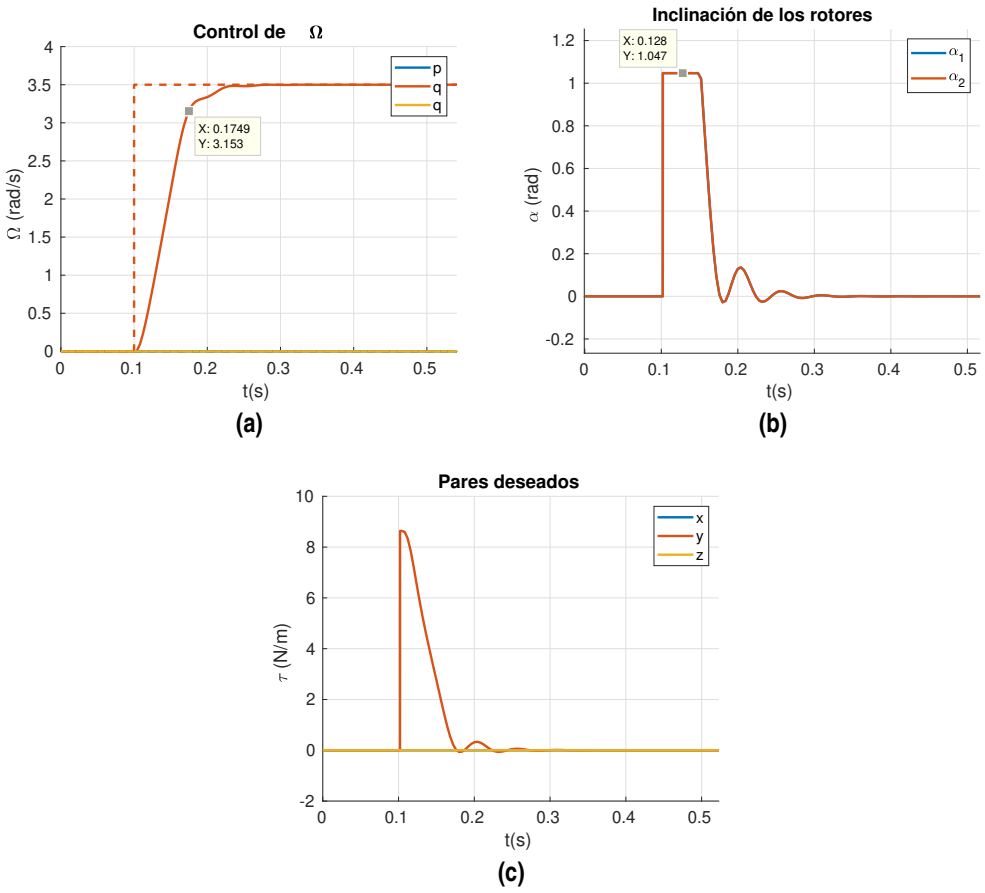

Figura 4.6 Experimento del controlador de velocidad angular.

Además, a partir de una frecuencia, el modelo de 4.19 comienza a ser menos exacto [3] .

$$t_s \approx \frac{\pi}{\omega_n} \tag{4.26}$$

$$t_s = 0.05s \rightarrow \omega_n = 62.83 rad/s \rightarrow K_p = 2.4674 \frac{N \cdot m}{rad/s} \tag{4.27}$$

$$\xi = 1 \rightarrow K_d = 0.0228 \frac{N \cdot m}{rad/s^2} \tag{4.28}$$

A parte de las ganancias, otros parámetros que influyen en en resultado son las saturaciones. Se ha elegido un ángulo máximo de inclinación de los servomotores (α_{max}) de 60 °. Si este valor es muy alto el vehículo podría perder altura, y si es muy bajo los pares que se aplican son menores.

El resultado del control con los parámetros calculados se puede observar en la figura 4.6. El tiempo de subida es más largo que el que se pidió en el diseño. Esto se debe a que los actuadores han saturado.

4.3.2 Control angular

Para seleccionar los parámetros de este controlador, se hará una simplificación de la ley de control de la ecuación 2.23 para que solo esté aplicada a un solo eje. Se va a tomar la estrategia de suponer que los 3 ejes se comportan de manera independiente en lugar de como un conjunto en forma de cuaternio. De esta manera, se podrá diseñar el controlador a partir de una función de transferencia.

[3] Esto tiene efecto sobretodo en las simulaciones con *PX4/Gazebo* ya que en ese escenario está presente el estimador de estados.

En el caso de que solo existiesen referencias en un solo eje, por ejemplo en el eje y, la orientación deseada expresada en cuaternios ($\boldsymbol{q}_{sp}$) en función del ángulo deseado en el eje y (θ_{sp}) sería la siguiente.

$$\boldsymbol{q}_{sp} = \begin{bmatrix} cos(\frac{\theta_{sp}}{2}) \\ 0 \\ sin(\frac{\theta_{sp}}{2}) \\ 0 \end{bmatrix} \tag{4.29}$$

Además, si suponemos que solo existe movimiento en el eje y del vehículo, la orientación estimada ($\boldsymbol{q}$) y el error ($\boldsymbol{q}_e$) quedan de la siguiente manera:

$$\boldsymbol{q} = \begin{bmatrix} cos(\frac{\theta}{2}) \\ 0 \\ sin(\frac{\theta}{2}) \\ 0 \end{bmatrix} \tag{4.30}$$

$$\boldsymbol{q}_e = \boldsymbol{q}^{-1} \cdot \boldsymbol{q}_{sp} = \begin{bmatrix} cos(\frac{\theta_{sp}}{2} - \frac{\theta}{2}) \\ 0 \\ sin(\frac{\theta_{sp}}{2} - \frac{\theta}{2}) \\ 0 \end{bmatrix} \tag{4.31}$$

Sustituyendo estas expresiones en la ecuación 2.23 la ley de control simplificada al eje y es la siguiente:

$$\boldsymbol{\Omega}_{sp} = \begin{bmatrix} p_{sp} \\ q_{sp} \\ r_{sp} \end{bmatrix} = 2\boldsymbol{K}_p * \begin{bmatrix} 0 \\ sin(\frac{\theta_{sp}}{2} - \frac{\theta}{2}) \\ 0 \end{bmatrix} \; sign\left(cos\left(\frac{\theta_{sp}}{2} - \frac{\theta}{2}\right)\right) \tag{4.32}$$

Para valores pequeños del error $\theta_{sp} - \theta$, la ley de control se puede aproximar:

$$q_{sp} = 2 \cdot K_p sen\left(\frac{\theta_{sp}}{2} - \frac{\theta}{2}\right) sign\left(cos\left(\frac{\theta_{sp}}{2} - \frac{\theta}{2}\right)\right) \approx K_p(\theta_{sp} - \theta) = K_p(\theta_e) \tag{4.33}$$

Esta ley de control sí se puede expresar en forma de función de transferencia:

$$C(s) = \frac{q_{sp}(s)}{\theta_e(s)} = K_p \tag{4.34}$$

El sistema a controlar sería el conjunto del controlador de velocidad angular más el tiltrotor, pero en lugar de tener en cuenta ese conjunto para el diseño, se va a realizar una aproximación que consiste en suponer que el controlador de velocidad angular es mucho más rápido que este y por tanto su función de transferencia de bucle cerrado se toma de la siguiente manera:

$$G_{inner}(s) = \frac{q_{sp}(s)}{q(s)} \approx 1 \tag{4.35}$$

Según [8], para que esta función de transferencia se aproxime a la real desde el punto de vista del controlador de ángulo, este tiene que ser diseñado con un ancho de banda al menos 5 veces menor que el del controlador interno. Utilizando esa hipótesis, la función de transferencia del sistema a controlar se va a suponer la siguiente:

$$G(s) = \frac{\theta(s)}{q_{sp}(s)} = \frac{1}{s} \tag{4.36}$$

Teniendo las funciones de transferencia del sistema y del controlador, se calcula la de bucle cerrado:

$$G_{BC}(s) = \frac{\theta(s)}{\theta_{sp}(s)} = \frac{G(s)K_p}{1 + G(s)K_p} = \frac{K_p}{s + K_p} = \frac{1}{s\frac{1}{K_p} + 1} \tag{4.37}$$

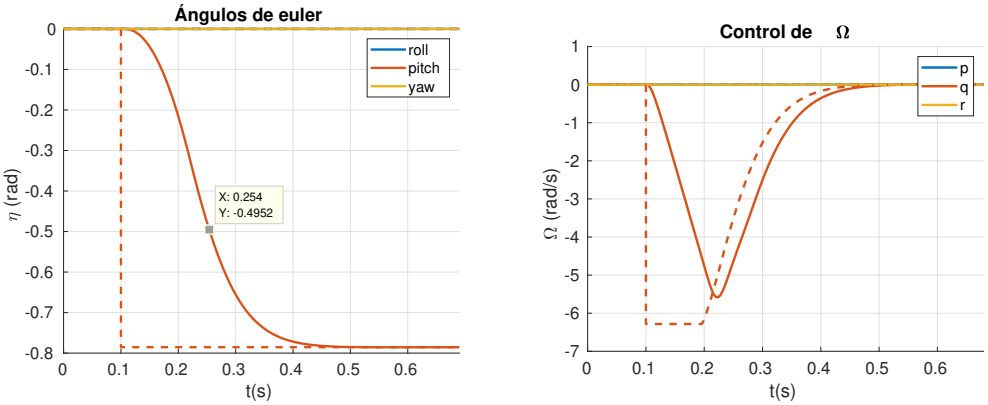

Figura 4.7 Experimento del controlador de orientación.

Para que se mantenga la hipótesis de que tenemos control directo sobre la velocidad angular, se va a pedir un ancho de banda 6 veces menor que el del controlador interno (ω_{n_inner}) .

$$\omega_n = \omega_{n\_inner}\frac{1}{6} = 12.57\ rad\frac{1}{6} = 10.4720\ rad \tag{4.38}$$

$$\tau = \frac{1}{\omega_n} = 0.0955s \rightarrow K_p = \frac{1}{\tau} = 10.4720\frac{rad/s}{rad} \tag{4.39}$$

El resultado puede verse en la figura 4.7.

4.3.3 Control de velocidad

Como se vio en la sección 2.3.2, este controlador genera unas fuerzas deseadas en ejes inerciales ($\boldsymbol{F}_{NED}$) a partir de los errores en velocidad. Para conseguir aplicar esas fuerzas en el vehículo, existe un módulo que genera una orientación deseada ($\boldsymbol{q}$) y una fuerza en el eje z del vehículo (F_z). Esa transformación se hace teniendo en cuenta que en los multirrotores solo aplican fuerza en su eje z, sin embargo, en el caso del tiltrotor, además de una fuerza en el eje z, también se genera una fuerza en el eje x (ver ecuación 4.6). Para valores pequeños de inclinación de los rotores (α_1 y α_2) esta fuerza es pequeña, y por tanto se dejará dicho módulo sin modificarlo y se despreciará esta diferencia del tiltrotor con respecto a los multirrotores habituales.

Si se supone que es posible generar fuerzas deseadas en los tres ejes del espacio ($\boldsymbol{F}_{NED}$), la ecuación dinámica del sistema a controlar es la siguiente:

$$\Sigma \boldsymbol{F} = m\,\boldsymbol{a} \tag{4.40}$$

$$\Sigma \boldsymbol{F} = \boldsymbol{F}_g + \boldsymbol{F}_{NED} \tag{4.41}$$

Esta se ha obtenido de la ecuación 3.11 sustituyendo las fuerzas que dependen de los actuadores por ($\boldsymbol{F}_{NED}$). En el dominio de Laplace para el eje x:

$$F_{x\_NED}(s) = m\,s\,V_x(s) \rightarrow \tag{4.42}$$

$$G_x(s) = \frac{V_x(s)}{F_{x\_NED}(s)} = \frac{1}{m\,s} \tag{4.43}$$

Además, se añadirá un polo a la función de transferencia para modelar la dinámica de los controladores internos, es decir, el seguimiento de las fuerzas deseadas. Esta dinámica se supondrá igual que la del seguimiento de ángulo, y se aproximará por una de primer orden con una constante de tiempo τ_F.

$$G(s) = \frac{V_x(s)}{F_{x\_NED}(s)} = \frac{1}{m\,s(\tau_F s + 1)} \tag{4.44}$$

Dicha constante de tiempo se hallará inspeccionando la respuesta ante escalón del controlador de orientación. Se puede comprobar en la figura 4.7 que la salida tarda en alcanzar el 63 % de la referencia en 0.15 s:

$$\tau_F = 0.15s \tag{4.45}$$

El controlador tendrá la misma estructura que el velocidad angular (figura 4.5), que es un PD con la acción derivativa aplicada únicamente a la medida. Dicho esto, la función de transferencia de bucle cerrado queda:

$$G_{BC}(s) = \frac{K_p G(s)}{1 + K_p G(s) + K_d G(s) s} = \frac{K_p}{m\tau_F s^2 + (m + K_d)s + K_p} = \frac{\frac{K_p}{m\tau_F}}{s^2 + \frac{(m+K_d)}{m\tau_F}s + \frac{K_p}{m\tau_F}} \tag{4.46}$$

Por comparación con un sistema de segundo orden donde ξ es el factor de amortiguamiento y ω_n la frecuencia de oscilación natural, se pueden hallar las ganancias del control:

$$G(s) = \frac{\omega_n^2}{\omega_n^2 + 2\omega_n \xi + s^2} \tag{4.47}$$

$$\omega_n = \sqrt{\frac{K_p}{m\tau_F}} \tag{4.48}$$

$$\xi = \frac{(m + K_d)}{2\omega_n m\tau_F} \tag{4.49}$$

Se despejan las dos ganancias del controlador:

$$K_p = \omega_n^2 m\tau_F \tag{4.50}$$

$$K_d = \xi 2\omega_n m\tau_F - m \tag{4.51}$$

Para que resulte un sistema críticamente amortiguado (dos polos reales e iguales) hacemos $\xi = 1$. Además se pedirá un ancho de banda 3 veces inferior al controlador de orientación.

$$\omega_n = \frac{\omega_{n\_inner}}{3} = \frac{10.472}{3} = 3.49 rad \rightarrow K_p = 1.8277 \frac{N}{m/s} \tag{4.52}$$

$$\xi = 1 \rightarrow K_d = 0.0472 \frac{N \cdot m}{rad/s^2} \frac{N}{m/s^2} \tag{4.53}$$

El resultado puede verse en la figura 4.8.

En cuanto a la velocidad en el eje z, existen dos fuerzas: la que depende del empuje que generan los rotores F_{z_NED} y la fuerza de gravedad $m\ g$:

$$F_{z\_NED} + m\ g = m\ \ddot{P}_z \tag{4.54}$$

F_{z_NED} es la señal de control e incluye un término constante para compensar la gravedad:

$$F_{z\_NED} = F_{z\_aux} - m\ g \tag{4.55}$$

Sustituyendo esta expresión en 4.54:

$$F_{z\_aux} = m\ \ddot{P}_z \tag{4.56}$$

En el dominio de Laplace:

$$F_{z\_aux}(s) = m\ s\ V_z(s) \rightarrow \tag{4.57}$$

$$G_z(s) = \frac{V_z(s)}{F_{z\_aux}(s)} = \frac{1}{m\ s} \tag{4.58}$$

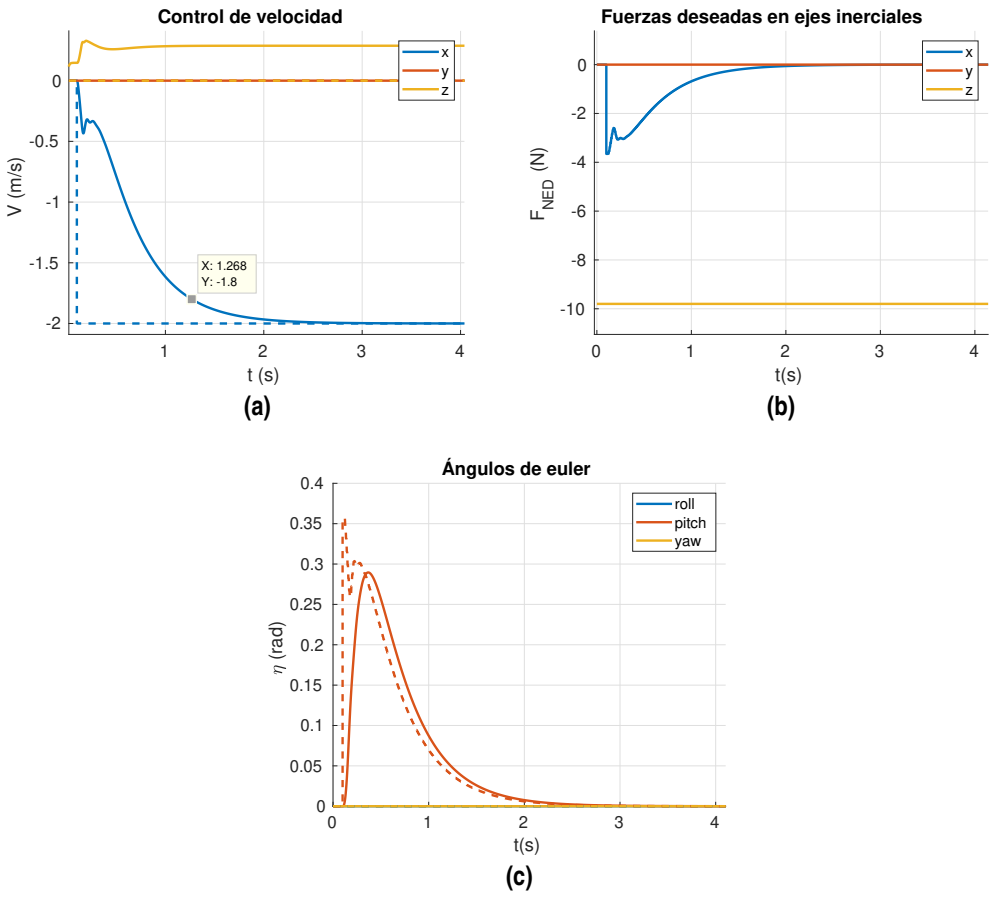

Figura 4.8 Experimento del controlador de velocidad.

No es posible aplicar cambios instantáneos de fuerza, así que se añadirá un polo que modela dinámica de la propulsión:

$$G_z(s) = \frac{V_z(s)}{F_{z\_aux}(s)} = \frac{1}{m\, s(\tau_m\, s+1)} \tag{4.59}$$

El controlador tiene la misma estructura que el de los ejes xy y se diseñará de la misma manera. Sin embargo, esta vez se pedirá un tiempo de subida más pequeño ($t_s = 0.1s$):

$$K_p = \omega_n^2 m\tau_m = \left(\frac{\pi}{0.1}\right)^2 m\ \tau_m = 39.4784\frac{N}{m/s} \tag{4.60}$$

$$K_d = \xi 2\frac{\pi}{0.1} m\tau_m - m = 0.2566\frac{N}{m/s^2} \tag{4.61}$$

El resultado se muestra en la figura 4.9

4.3.4 Control de posición

Este controlador genera velocidades de referencia ($\boldsymbol{V}_{sp}$) a partir de errores de posición. Al igual que el controlador de ángulo, este es de tipo proporcional y también se supondrá que la dinámica de la señal de actuación (G_{inner}) es nula, es decir, la velocidad se controla directamente.

$$G_{inner}(s) = \frac{V(s)}{V_{sp}(s)} \approx 1 \rightarrow G(s) = \frac{P(s)}{V(s)} = \frac{1}{s} \tag{4.62}$$

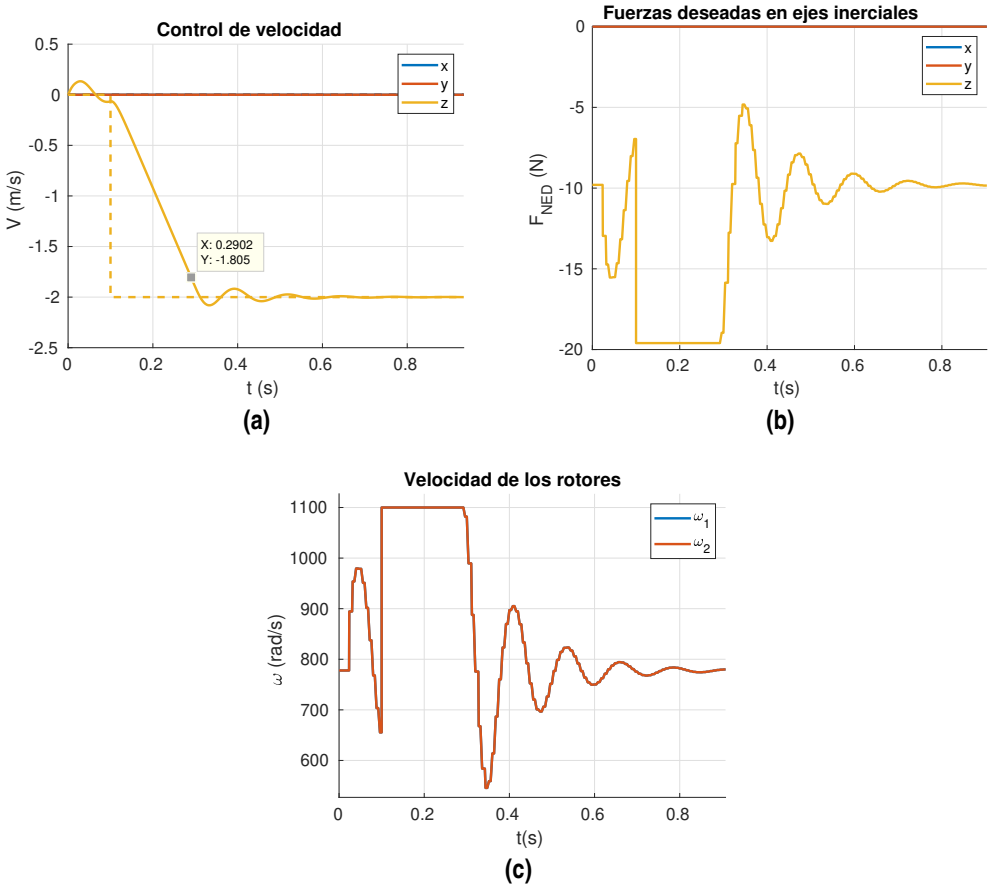

Figura 4.9 Experimento del controlador de velocidad en el eje z.

La función de transferencia del controlador y de bucle cerrado son las siguientes:

$$C(s) = K_p \tag{4.63}$$

$$G_{BC}(s) = \frac{P(s)}{P_{sp}(s)} = \frac{G(s)K_p}{1+G(s)K_p} = \frac{K_p}{s+K_p} = \frac{1}{s\frac{1}{K_p}+1} \tag{4.64}$$

Para que dicha simplificación sea aplicable, se va a pedir un ancho de banda 5 veces inferior al del controlador de velocidad:

$$\omega_n = \omega_{inner}\frac{1}{5} = 3.4907\ rad\frac{1}{5} = 0.6981 rad \tag{4.65}$$

$$\tau = \frac{1}{\omega_n} = 1.4324s \rightarrow K_p = 0.6981\frac{m/s}{m} \tag{4.66}$$

El resultado puede verse en la figura 4.10. Se ve que el vehículo pierde altura porque las ganancias del controlador de velocidad en el eje z son nulas en este experimento.

4.4 Diseño mediante optimización numérica

En algunas ocasiones, el diseño analítico de un controlador se realiza a partir de un modelo matemático que no describe de manera muy correcta el sistema real en todas las condiciones de operación. Esto puede deberse a las no linealidades del sistema como por ejemplo las saturaciones del actuador. Por esa razón, en esta sección, para diseñar el controlador se va a utilizar un método iterativo que minimiza una función de coste (optimización). En la función de coste se realizan dos tareas: primero se ejecuta un experimento con un modelo de *Simulink*, en el que se aplica un escalón a la referencia de alguno de los controladores

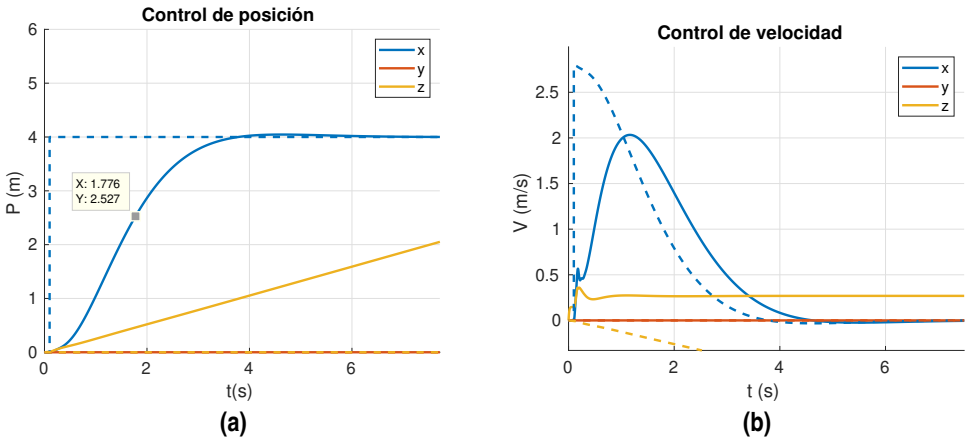

Figura 4.10 Experimento del controlador posición.

y una perturbación, y segundo, a partir de los resultados se calcula un índice que indica el desempeño del controlador.

Dicho modelo de *Simulink* es el mismo con el que se ha probado los controladores obtenidos mediante el diseño analítico. El modelo del tiltrotor consiste en las ecuaciones halladas en la sección 3.1 que son más precisas que los modelos utilizados para el diseño analítico de los controladores (4.19, 4.36, 4.44, 4.62).

Para la optimización, se hará uso de una función de *Matlab* llamada *fmincon* que permite estimar el mínimo local de funciones con múltiples parámetros. Su uso pude verse en el siguiente código.

Código 4.4.1: Script en el que se realiza la optimización (*optimizar.m*)

```
clear all;clc
parametros; % Script que da valor a la estructura global 'param'. Esta almacena
  ↪ parámetros como las ganancias de los controladores o el tiempo de simulación
modo_control(param.modoControl);  % Función que modifica el estado de los switches del
  ↪ modelo de simulink para desactivar algunos controladores y activar otros
addpath('optimizacion') % Carpeta en la que se encuentran algunas funciones de utilidad

% X = fmincon(FUN,X0,A,B,Aeq,Beq,LB,UB,NONLCON,OPTIONS)
%       -FUN: Función a optimizar. Como entrada tiene un solo vector de tamaño
%      el número de parámetros.
%       -X0:  Vector de estados iniciales
%       -A,B,Aeq,Beq: Matrices para establecer los límites (no se utilizan)
%       -X0:  Vector de estados iniciales
%       -LB:  Límite inferior de los parámetros
%       -UB:  Límite superior de los parámetros
%       -NONLCON: Función que establece límites (no se utiliza)
%       -OPTIONS: estrutura de datos en la que se indican las opciones,
%      como por ejemplo el método.

options =optimoptions('fmincon','Display','iter','Algorithm','interior-point'); % Se pide
  ↪ que se imprima por pantalla cada vez que se encuentre un mínimo y se elige el método
  ↪ de optimización 'punto interior'. Este es un algoritmo de gran escala el cual se
  ↪ recomienda para utilizarlo a la hora de enfrentarse a un problema nuevo.

x0=[0.6981 ,  0, 0]; % Parámetros iniciales
lb=[0,0,0]; % Límite inferior de los parámetros
ub=[30,0,0]; % Límite superior de los parámetros


tic % Funciones tic y toc para evaluar tiempo de ejecución la función.

x=fmincon(@fun_op,x0,[],[],[],[],lb,ub,[],options)% Comienzo de la optimización

toc
```

Como se indica en los comentarios, la función a optimizar se llama *fun_op*. Esta recibe 3 argumentos que corresponden a las ganancias proporcional, integral y derivativa del controlador que se está diseñando. Los

estados iniciales para la optimización serán las ganancias obtenidas en el diseño analítico. Estos estados pueden tener efecto en el resultado de la optimización ya que la función *fmincon* no busca mínimos globales, sino locales.

La función *fun_op* se encarga de realizar el experimento con las ganancias K_p, K_i y K_d que se dan como entradas, recoger sus resultados y devolver como resultado el coste.

Código 4.4.2: Función a optimizar (*fun_op.m*)

```
function out = fun_op(x)
    Kp=x(1)
    Ki=x(2)
    Kd=x(3)
    establece_ganancias(Kp,Ki,Kd); % Función que modifica las ganancias del controlador
    ↪  que se esté optimizando (el de velocidad angular, orientación...) y del eje que se
    ↪  está probando (x, y o z)
    datos = sim_sk(); % Se ejecuta la simulación y se guarda los resultados en el objeto
    ↪  'datos'
    [y,r,t] = obtiene_resultados(datos); % Se extraen la medida (y), la referencia (r) y
    ↪  de tiempo (t) del objeto 'datos'.
    J=Coste(r,y,t); % Se calcula el coste
    out=J;
end
```

Algunos criterios usados para calcular el coste son [9]:

- IAE (*Integral Absolute Error*) $= \int_0^t \mid e(t) \mid dt$
- ISE (*Integral Square Error*) $= \int_0^t e(t)^2 dt$
- ITAE (*Integral Time Absolute Error*) $= \int_0^t t \mid e(t) \mid dt$
- ITSE (*Integral Time Square Error*) $= \int_0^t t\ e(t)^2 dt$

El criterio que se va a utilizar va a ser el de la integral del error absoluto(IAE) La implementación en *Matlab* se ha realizado con una integral trapezoidal:

Código 4.4.3: Funcion de coste (*Coste.m*)

```
  function int = Coste(r,y,t)
% Función que calcula el coste
    % Entradas:
    % - r: vector de referencias
    % - y: vector de salidas
    % - t: vector de tiempos

    e=abs(r-y); % Se calcula el error valor absoluto del error
    int=0;
    for i=2:length(t)
            int=int+0.5*(e(i)+e(i-1)); % integral trapezoidal
    end
end
```

Se comenzará la optimización por el controlador de velocidad angular. A diferencia del diseño analítico, se le aplicará una perturbación en la mitad del tiempo de simulación. En la figura 4.11 se puede ver el desempeño de lo controlador de velocidad angular con una perturbación de $\tau_y = 0.5Nm$ en el instante 0.5s.

Utilizando las ganancias obtenidas en la optimización del controlador anterior, se vuelve a ejecutar el proceso pero con un escalón en la referencia del ángulo θ. En la figura 4.12 se muestra la respuesta temporal con el controlador obtenido aplicando la misma perturbación que en el caso anterior.

El controlador que está en un nivel superior es el de velocidad, el cual se optimizará con la respuesta a un escalón en la referencia de 2 m/s y una perturbación de una fuerza de 0.2 N sobre el eje x en el instante $t = 4s$ (ver figura 4.13). Se puede ver que este controlador produce una salida más oscilatoria que el obtenido mediante diseño analítico, ya que este comportamiento no aumenta el coste con el criterio elegido. Además, su coste es menor debido a un transitorio más rápido y a que en la zona de la perturbación el coste con el controlador analítico es mayor.

Por último, se halla la ganancia proporcional del controlador de posición, con la que se tiene la respuesta mostrada en la figura 4.14. En la tabla 4.1 se puede ver que esta ganancia es muy parecida a la obtenida en

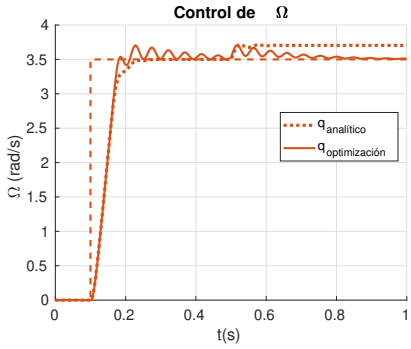

Figura 4.11 Comparación de los resultados de los controladores obtenidos por diseño analítico e iterativo en el seguimiento y en el rechazo a perturbaciones.

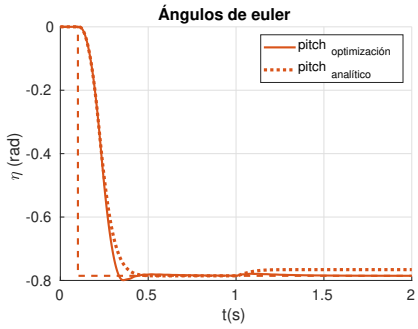

Figura 4.12 Comparación de los controladores de orientación.

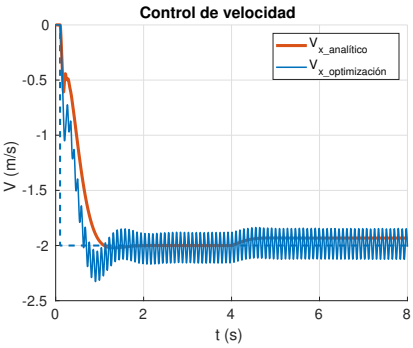

Figura 4.13 Comparación de los controladores de velocidad.

el método analítico (ocurre de forma casual) pero producen resultados diferentes porque los controladores interiores son diferentes.

	Optimización			Analítico		
	K_p	K_i	K_d	K_p	K_i	K_d
Control de vel. angular	4.2473	22.5534	0.0343	2.4674	0	0.0228
Control de angular	12.4518	-	-	10.4720	-	-
Control de velocidad	7.3701	0.4456	0.1202	2.9588	0	0.0472
Control de posición	0.69810	-	-	0.69813	-	-

Tabla 4.1 Ganancias obtenidas. Las unidades corresponden a las que se mostraron en la sección 4.3.

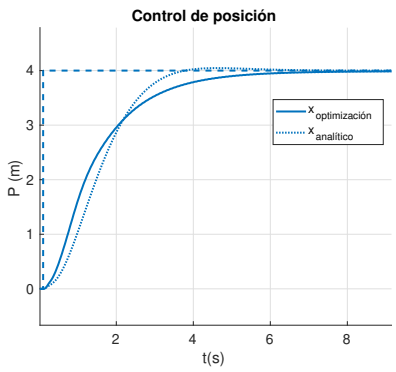

Figura 4.14 Comparación de los controladores de posición.

5 Simulación *PX4/Gazebo*

Gazebo es un simulador robótico de código abierto, que entre sus funcionalidades destacan la representación en 3D y la inclusión de sensores, tales como LIDARs. Posee una interfaz gráfica en la que se puede crear y modificar la simulación, pero para utilizar todo su potencial, es necesario editar o crear ciertos archivos de texto:

- Archivos de descripción (.sdf). Su lenguaje es el XML y con ellos se describe el robot y su entorno. Por ejemplo, la masa o la inercia de los sólidos, los enlaces entre ellos (rotación, translación...), los elementos del escenario, los plugins utilizados, etc.
- Plugins (.cpp). Estos son los que permiten personalizar al máximo la simulación, ya que en ellos se puede utilizar la API de *Gazebo*. Entre otras cosas, permiten aplicar fuerzas y pares a los enlaces y sólidos, comunicarse con otros programas mediante sockets, agregar elementos visuales sin dinámica como líneas, etc.

Otra característica que tiene es que es capaz de ejecutar la simulación en tiempo real, algo que es deseable cuando se realizan simulaciones *SITL* (Software-In-The-Loop) o *HITL* (Hardware-In-The-Loop). Ambas son técnicas para el desarrollo de controladores o sistemas embebidos que permiten probar su funcionamiento sin utilizar una planta real (en este caso un tiltrotor) sino una simulada. En ambos casos se utiliza un modelo matemático que desde el punto de vista del controlador se comporta de manera parecida a la planta real. La diferencia entre ambos métodos está en dónde se ejecuta el software sistema embebido (el autopiloto en el caso de este trabajo): en el caso de *HITL* se ejecuta en el hardware real (por ejemplo en el microcontrolador *STM32F427* que se mencionó al principio del capítulo), mientras que en una simulación *SITL* se ejecuta en la misma máquina que el modelo de la planta, por ejemplo en un ordenador con sistema operativo Ubuntu.

Se pueden realizar ambas simulaciones con *PX4* pero en este trabajo se va a utilizar *SITL*. En ella el autopiloto se ejecuta en un proceso con el mismo código que el autopiloto real (salvo ciertas partes que sirven para comunicarse con el simulador) y el simulador se ejecuta en otro proceso. Como se ve en la figura 5.1, ambos procesos se comunican mediante protocolos de la capa de transporte de internet (UDP y TCP), lo que podría permitir que se ejecutasen en máquinas distintas si la latencia no fuese demasiado alta. Se puede ver que también existe la GCS, que permite interactuar con el autopiloto exactamente igual que si se tratase de un vuelo real (establecer waypoints, cambiar el modo de vuelo, modificar los parámetros, conectar un mando...). Por último, también podría estar presente cualquier otra aplicación (en la figura denominada *API/Offboard*) que utilice el protocolo *MAVLink* , como por ejemplo un nodo de ROS, una aplicación escrita en python, etc.

En cuanto a la simulación, existen plugins especialmente creados para este autopiloto, ya que por defecto Gazebo no trae algunas funcionalidades como la inclusión de rotores. Algunos de los utilizados son:

- *gazebo_mavlink_interface.cpp*. Implementa la comunicación con el autopiloto mediante sockets. Envía las lecturas de los sensores (IMU, GPS, flujo óptico...), y recibe las señales de los actuadores generadas por el autopiloto.
- *gazebo_wind_plugin.cpp*. Aplica fuerzas al vehículo en función de la velocidad del mismo.
- *gazebo_imu_plugin.cpp*. Simula las lecturas de un una IMU añadiendo ruido a las medidas.
- *gazebo_motor_model.cpp*. A partir de una velocidad de giro deseada, aplica una fuerza y un par al cuerpo del vehículo con una dinámica de primer orden (simulando al conjunto ESC-motor). Además, hace girar el rotor.

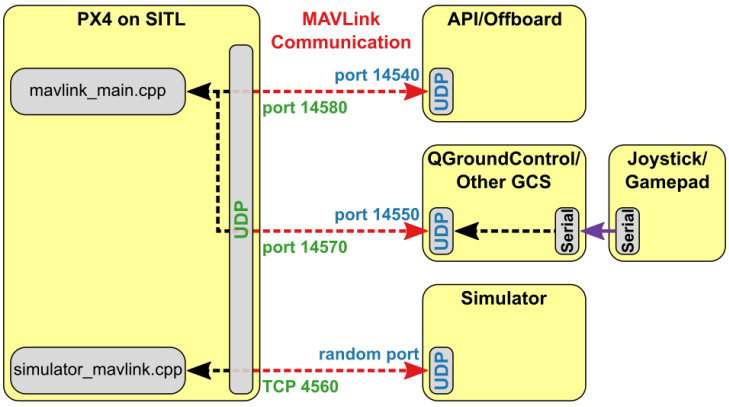

Figura 5.1 Vista general de la simulación [Fuente: dev.px4.io].

- *liftdrag_plugin.cpp*. Genera fuerzas y pares que modelan una superficie aerodinámica.

Todos los parámetros utilizados por los plugins (coeficiente de empuje de los rotores, offset de error en los sensores, coeficiente de rozamiento aerodinámico, constante de tiempo de los motores...) se encuentran en el archivo de descripción con la extensión *.sdf*. Para el caso de un quadrotor, por ejemplo, además de estos parámetros, se establece que existen 5 sólidos (4 rotores y el cuerpo principal) y 4 articulaciones de rotación.

5.1 Cambios de la simulación en *Gazebo*

Existen varios vehículos disponibles para la simulación con *Gazebo* realizados por la comunidad de *PX4* (quadrotor, avión, submarino, rover...), sin embargo, no existía ningún tiltrotor de dos rotores, por tanto, ha tenido que ser creado. Como se ha visto en la anterior sección, las simulaciones están compuestas por vehículos multicuerpo y de plugins, que entre otras funciones, añaden ruido a las medidas. En este trabajo se ha querido simplificar la simulación y se ha supuesto que el vehículo está compuesto por un solo sólido rígido al que se le pueden aplicar dos fuerzas orientables.

Uno de los aspectos que hay que modificar es cómo se generan las fuerzas en función de las señales de actuación. El cálculo y aplicación de dichas fuerzas se realiza en la siguiente función [1] :

Código 5.1.1: Función donde se aplican las fuerzas al tiltrotor (*gazebo_mavlink_interface.cpp*)

```
1154  void GazeboMavlinkInterface::handle_control_2(double _dt)
1155  {
1156  /*
1157   *Entradas:
1158   * - 'input_reference': Vector de 4 componentes que indica las señales de
1159   * actuación normalizadas. Su rango es (0,1). Es un miembro de la clase
1160   * y por eso no es necesario incluirlo como argumento de entrada.
1161   * - '_dt': Diferencia de tiempo entre las llamadas a esta función. Este se
1162   * usa para aplicar filtros.
1163   */
1164
1165    /*** Parámetros del tiltrotor ***/
1166    double ct=8.0992e-6; // Coeficiente de empuje [N/(rad/s)^2]
1167    double l_z=0.25; // Distancia en el eje z desde el c.m. a los rotores [m]
1168    double l_y=0.5; // Distancia en el eje y desde el c.m. a los rotores [m]
1169    double omega_max=1100; // Máxima velocidad de rotación de los rotores [rad/s]
1170    double cm=0; // Coeficiente de arrastre
1171
1172    double omega1=input_reference_[0]*omega_max;  // Velocidad angular del rotor 1 (rango
      ↪  0,1100) [rad/s]
1173    double omega2=input_reference_[1]*omega_max; // Velocidad angular del rotor 2 (rango
      ↪  0,1100) [rad/s]
```

[1] Esta se encuentra en el plugin de comunicación *gazebo_mavlink_interface.cpp* y sustituye a la función original *handle_control()*.

```
1174     double alfa1=(input_reference_[2]-0.5)*2*M_PI/2; // Ángulo del rotor 1 (rango
         ↪  -pi/2,pi/2) [rad]
1175     double alfa2=(input_reference_[3]-0.5)*2*M_PI/2; // Ángulo del rotor 2 (rango
         ↪  -pi/2,pi/2) [rad]
1176
1177     // Filtrado de la velocidad ángular de los motores con una dinámica de
1178     // primer orden con una constante de tiempo de 0.01 s. Esto modela la
1179     // dinámica del conjunto ESC-motores.
1180     omega1=filtro_motor1.updateFilter(omega1,_dt);
1181     omega2=filtro_motor2.updateFilter(omega2,_dt);
1182
1183     // Se añade una dinámica de primer orden a los servomotores. También con una
1184     // constante de tiempo de 0.01s
1185     alfa1=filtro_servo1.updateFilter(alfa1,_dt);
1186     alfa2=filtro_servo2.updateFilter(alfa2,_dt);
1187
1188     // Obtención de el empuje de los rotores.
1189     double T1=omega1*omega1*ct; // [N]
1190     double T2=omega2*omega2*ct; // [N]
1191
1192     // Posición de los dos rotores. Hay que tener en cuenta que en Gazebo el
1193     // eje z apunta hacia arriba, al contrario que lo supuesto en las
1194     // ecuaciones
1195     math::Vector3 p1(0, -l_y,l_z);
1196     math::Vector3 p2(0,l_y,l_z);
1197
1198     // Vector de fuerza del rotor 1. Este se halla teniendo en cuenta que
1199     // gira alrededor del eje y, y por tanto, la componente en dicho eje será
1200     // constante y nula.
1201     double Fx1=-T1*sin(alfa1); // Componente en el eje x
1202     double Fz1=T1*cos(alfa1); // Componente en el eje z
1203     math::Vector3 f1(Fx1,0,Fz1); // Vector de fuerza del rotor 1
1204
1205     double Fx2=-T2*sin(alfa2); // Componente en el eje x
1206     double Fz2=T2*cos(alfa2); // Componente en el eje z
1207     math::Vector3 f2(Fx2,0,Fz2); // Vector de fuerza del rotor 2
1208
1209     /*** Aplicación de los empujes ***/
1210     // Se hace uso la siguiente función perteneciente a la API de Gazebo:
1211     link_->AddLinkForce(f1,p1); // Aplica la fuerza f1 en la posición p1. Ambos expresados
         ↪  en ejes del cuerpo
1212     link_->AddLinkForce(f2,p2); // Aplica la fuerza f2 en la posición p2. Ambos expresados
         ↪  en ejes del cuerpo
1213     alfa1=filtro_servo1.updateFilter(alfa1,_dt);
1214     alfa2=filtro_servo2.updateFilter(alfa2,_dt);
1215
1216
1217     /*** Aplicación de los pares de reacción ***/
1218     double taud1=cm*omega1*omega1; // Par generado por el rotor 1
1219     double taud2=-cm*omega2*omega2; // Par generado por el rotor 2
1220     math::Vector3 Taud1(taud1*sin(alfa1),0, taud1*cos(alfa1)); // Vector de par 1
1221     math::Vector3 Taud2(taud2*sin(alfa2),0, taud2*cos(alfa2)); // Vector de par 2
1222     link_->AddRelativeTorque(Taud1); // Se aplica el par 1 expresados en ejes cuerpo
1223     link_->AddRelativeTorque(Taud2); // Se aplica el par 2 expresados en ejes cuerpo
1224
1225     // Por último, se envia un mensaje a un plugin de representación gráfica.
1226     // Este se encarga de graficar unas rectas para saber la magnitud de los
1227     // empujes y sus orientaciones
1228     gazebo::msgs::Quaternion datos;
1229     datos.set_x(Fx1);
1230     datos.set_y(Fz1);
1231     datos.set_z(Fx2);
1232     datos.set_w(Fz2);
1233     linea_pub_->Publish(datos);
1234   }
```

También es necesario definir los parámetros dinámicos del cuerpo (masas, inercias...) y los plugins utilizados. Esto se realiza en un archivo de descripción de extensión *.sdf*. Algunos fragmentos de dicho archivo se muestra a continuación:

Código 5.1.2: Archivo de descripción del tiltrotor (*iris.sdf*)

Se definen la masa y las inercias del único cuerpo que existe:

```
1   <sdf version='1.6'>
2     <model name='iris'>
3       <link name='base_link'>
4         <pose frame=''>0 0 0 0 -0 0</pose>
5         <inertial>
6           <pose frame=''>0 0 0 0 0 0</pose>
7           <mass>1</mass>
8           <inertia>
9             <ixx>0.03</ixx>
10            <ixy>0</ixy>
11            <ixz>0</ixz>
12            <iyy>0.04</iyy>
13            <iyz>0</iyz>
14            <izz>0.09</izz>
15          </inertia>
16        </inertial>
```

El modelo de colisión será una ortoedro con dimensiones parecidas a las del vehículo:

```
17        <collision name='base_link_inertia_collision'>
18          <pose frame=''>0 0 0 0 -0 0</pose>
19          <geometry>
20            <box>
21              <size>1 1.29 0.4</size>
22            </box>
23          </geometry>
```

La representación visual se obtiene de un archivo *.stl*:

```
39        <visual name='base_link_inertia_visual'>
40          <plugin name="visual_plugin" filename="libvisual_plugin.so"/>
41          <pose frame=''>0.60 0 0 0 -0 3.14</pose>
42          <geometry>
43            <mesh>
44              <scale>0.001 0.001 0.001</scale>
45              <uri>model://iris/tilt-rotor.stl</uri>
46            </mesh>
47          </geometry>
```

Se incluyen los plugins:

```
59      <plugin name='gps_plugin' filename='libgazebo_gps_plugin.so'>

63      <plugin name='mavlink_interface' filename='libgazebo_mavlink_interface.so'>

140     <plugin name='rotors_gazebo_imu_plugin' filename='libgazebo_imu_plugin.so'>
```

Los plugins utilizados han sido modificados de forma que no generen ruidos en las medidas, ya que se pretende evaluar los controladores suponiendo medidas perfectas.

El archivo *.stl* que se indica en la línea 45 alberga el modelo de un VTOL mostrado en la figura 5.2, cuya forma no afecta de ninguna manera en la simulación, ya que las colisiones se han definido como un ortoedro. De dicho modelo salen dos líneas moradas cuya longitud es proporcional a la velocidad de giro de los rotores y su orientación coincide con la inclinación de los motores. Para ello se ha creado un plugin, que se incluye en la línea 40 del archivo *.sdf*, y que hace uso de la clase *DynmicsLines*[2] de la API de *Gazebo*.

5.2 Entorno de simulación

Para comenzar la simulación basta con ejecutar el comando "*make px4_sitl gazebo_tiltrotor_birotor*" en la carpeta raiz del repositorio, abirendose *Gazebo* con el modelo elegido. En el terminal en el que se ejecutó el comando (parte superior izquierda de la figura 5.2) se puede interactuar con el piloto. Por ejemplo, el

[2] Documentada en http://osrf-distributions.s3.amazonaws.com/gazebo/api/7.1.0/classgazebo_1_1rendering_1_1DynamicLines.html

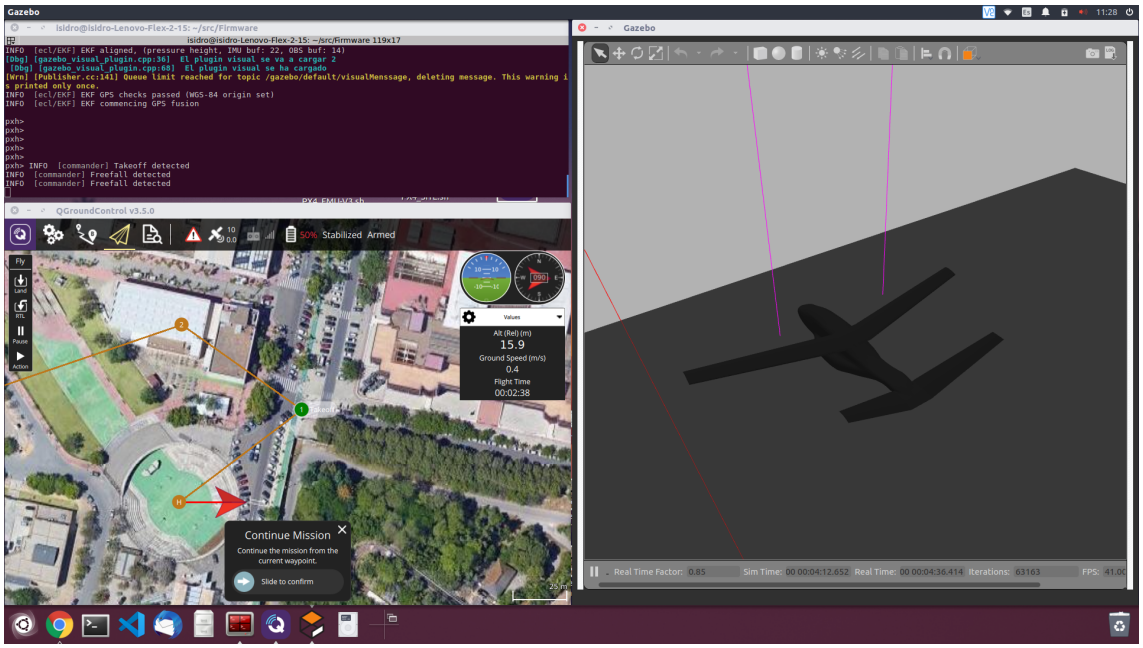

Figura 5.2 Entorno de simulación.

Figura 5.3 Mando *Logitech F310* para controlar el vehículo en la simulación. [Fuente: *amazon.es*].

comando "*listener actuator_outputs*" muestra el contenido del último mensaje de tipo *actuator_outputs* (indica el ancho de pulso en microsegundos que se le manda a los actuadores) que se publicó. Otro ejemplo es el comando "*commander mode acro*" el cual activa el modo *acro*. Una lista de todos los comandos admitidos está en la sección *Modules & Commands*[3] de la guia de desarrolladores de *PX4*. Además, a través de este terminal se puede realizar la depuración del programa[4].

Dicha manera de interactuar con el autopiloto no se ha usado mucho y en su lugar se ha utilzado la GCS *QGroundControl*, que se muestra en la parte inferior izquierda de la figura 5.2. Esta se conecta al autopiloto mediante sockets sin necesidad de realizar ninguna configuración. A modo de emisora se utilizará el mando *Logitech F310* (figura 5.3).

Entre las funcionalidades de *QGroundControl* está la modificación de parámetros en tiempo de vuelo (figura 5.4) como por ejemplo las ganancia de los PIDs. Por otro lado, hay parámetros que para que tengan efecto es necesario reiniciar el autopiloto como por ejemplo *EKF2_HGT_MODE* que determina el sensor que usa el estimador como medida de altura.

En la figura 5.5 se ve como se establecen los waypoints antes de comenzar una misión. La zona verde indica una zona que va a ser barrida (*survey*), la circunferencia es límite por del cual no se saldrá (*fence*) y el punto *R* es un *rally point*, el cual es un lugar en el que se podrá aterrizar alternativo al lugar de despegue.

Si se hace click en *Widgets>Analyze* se abre la ventana de la figura 5.6 donde muestra una representación temporal de algunos mensajes que se reciben del autopiloto.

[3] Ubicada en el siguiente enlace: https://dev.px4.io/v1.9.0/en/middleware/modules_main.html

[4] La información para iniciar la depuración se encuentra en https://dev.px4.io/v1.9.0/en/debug/simulation_debugging.html

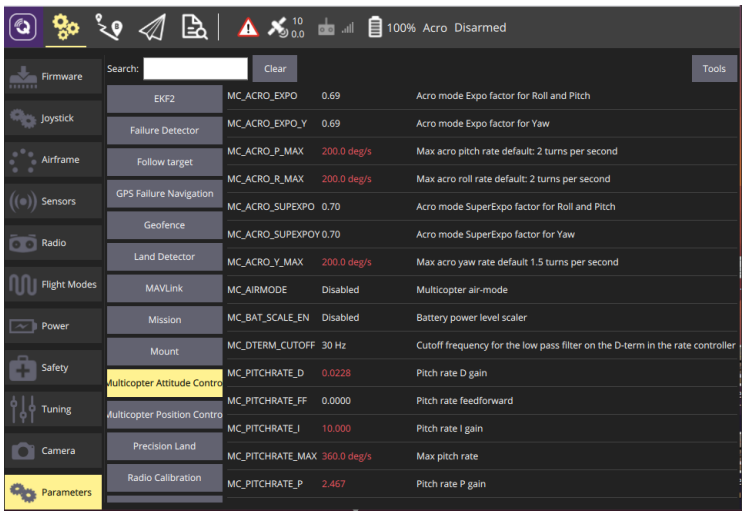

Figura 5.4 Ventana donde se modfican los parámetros.

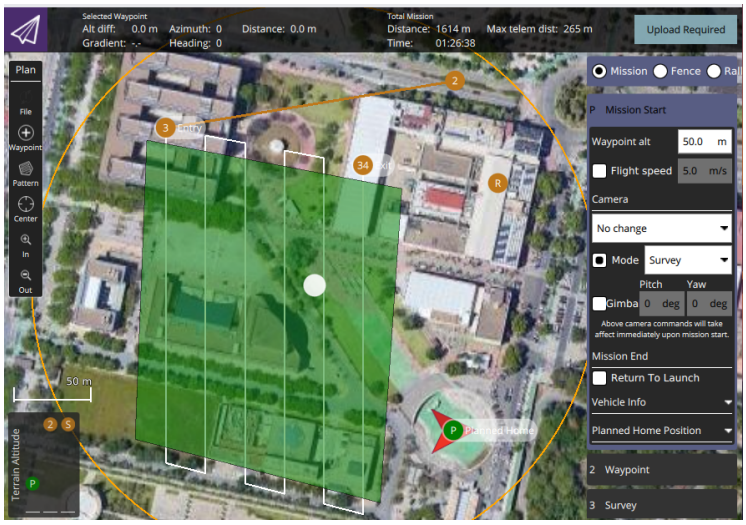

Figura 5.5 Generando una misión.

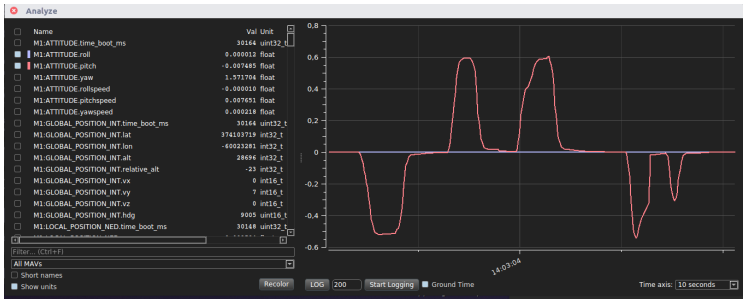

Figura 5.6 Ventana *Analyze*.

5.3 Cambios en *PX4*

Como ya se comentó en la sección 2.3.6, en *PX4* existen 3 categorías de mixer, y ninguna de ellas nos sirven para aplicar las ecuaciones halladas en 4.1 ya que estas no son lineales. Por tanto, se ha tenido que **crear un cnntrol *mixer*** diferente. Para cada categoría de mixer, existe una clase, las cuales están definidas en archivos separados en la carpeta Firmware/src/lib/mixer/. Uno de esos archivos es *mixer_multirotor.cpp*, que describe la categoría de los mixer multirrotor y que se ha usado como plantilla para la creación de uno nuevo. Para

adaptarlo, no ha sido necesario cambiar muchas partes del código, prácticamente solo se ha tenido que crear una función llamada *mix_tiltrotor* que se muestra a continuación ampliamente comentada:

Código 5.3.1: Fichero donde se describe el mixer para el tiltrotor (*mixer_tiltrotor.cpp*)

```
544 void TiltrotorMixer::mix_tiltrotor(float tau_x, float tau_y, float tau_z, float thrust,
     ↪  float *outputs)
545 {
546
547    /*
548     *Entradas:
549     *  - tau_x: Par deseado en el eje x del vehículo [N.m]
550     *  - tau_y: Par deseado en el eje y del vehículo [N.m]
551     *  - tau_z: Par deseado en el eje z del vehículo [N.m]
552     *  - *outputs: Dirección de memoria donde se guardarán las señales de los actuadores
     ↪  normalizadas (rango 0-1). En este caso hay 4 señales de actuación y por lo tanto
     ↪  será un vector de tamaño 4.
553     */
554
555    float pi=M_PI; // Número pi
556
557    /*** Parámetros del tiltrotor ***/
558    float lz=0.25; // distancia en el eje z desde los rotores al centro de gravedad [m]
559    float ly=0.5; // distancia en el eje y desde los rotores al centro de gravedad [m]
560    float cT=8.0992e-06; // Coeficiente de empuje de los rotores [N/(rad/s)^2]
561    float omega_max=1100; // Velocidad máxima de los rotores [rad/s]
562    float alfa_max=pi/2; // Inclinación máxima de los rotores [rad]
563
564    /*** Cálculo de los dos empujes T1 y T2 ***/
565    float T2=(tau_x+ly*thrust)/(2*ly); // Empuje del rotor 2 [N]
566    float T1=thrust-T2; // Empuje del rotor 1 [N]
567    if (T1<0) T1=0; // No se pueden generar empujes negativos ya que el rotor solo gira en
     ↪  un sentido
568    if (T2<0) T2=0;
569
570    /*** Cálculo de los dos ángulos alfa1 y alfa2 ***/
571    float alfa1=0; // Ángulo de inclinación del rotor 1 [rad]
572    float alfa2=0; // Ángulo de inclinación del rotor 2 [rad]
573    if (T1<float(0.00001)) { // Si el empuje del rotor 1 es muy pequeño, no se intenta
     ↪  controlar el par en el eje y ni en el eje z.
574       alfa1=0; // Se deja el rotor en vertical
575    }
576    else {
577       alfa1= 1/(2*lz*T1)*tau_y  + 1/(2*ly*T1)*tau_z;
578       alfa1= math::constrain(alfa1,-alfa_max, alfa_max); // Saturar alfa 1 entre los
     ↪  límites [-alfa_max, alfa_max]
579    }
580
581    if (T2<float(0.00001)) {
582       alfa2=0;
583    }
584    else {
585       alfa2= 1/(2*lz*T2)*tau_y  - 1/(2*ly*T2)*tau_z;
586       alfa2= math::constrain(alfa2,-alfa_max, alfa_max);
587    }
588
589
590    float omega1=sqrt(T1/cT); // Velocidad ángular del rotor 1 [rad/s]. Obtenido a partir
     ↪  de la ecuación que modela el empuje del rotor: T=cT*omega^2
591    float omega2=sqrt(T2/cT); // Velocidad ángular del rotor 2 [rad/s].
592
593
594    /*** Cálculo de las señales de actuación normalizadas (rango 0-1) ***/
595    outputs[0] = omega1/omega_max;
596    outputs[1] = omega2/omega_max;
597    outputs[2] = (alfa1+pi/2)/(pi);
598    outputs[3] = (alfa2+pi/2)/(pi);
599 }
```

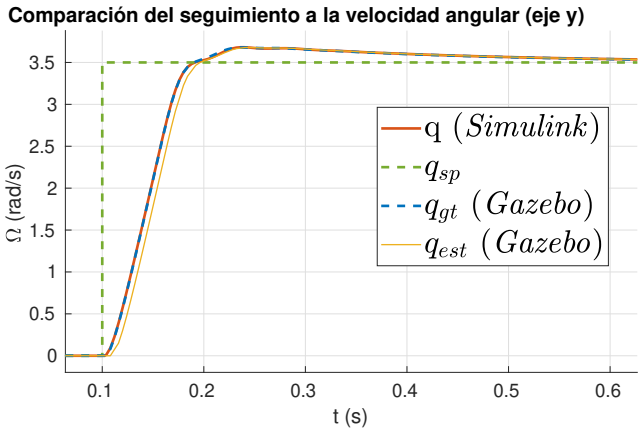

Figura 5.7 Comparación del seguimiento de velocidad angular. La ganancia integral es $K_i = 10\frac{N \cdot m}{rad}$.

Por último, solo es necesario [5] incluir el prototipo de la clase en el fichero de cabecera *mixer.h* e incluirlo en la compilación modificando el archivo *CMakeLists.txt*:

Código 5.3.2: Archivo para el programa CMake (*CMakeLists.txt*)

```
87  add_library(mixer
88          mixer.cpp
89          mixer_group.cpp
90          mixer_helicopter.cpp
91          mixer_load.c
92          mixer_multirotor.cpp
93          mixer_simple.cpp
94          mixer_tiltrotor.cpp # Se ha añadido esta línea
95  )
```

Originalmente las fuerzas y pares toman un valor dentro del rango de 0 a 1. Esto es así porque están saturadas, y las matrices de generación de señales de actuación que hay en el mixer, están hechas con ese objetivo, en vez de ser un modelo fiel del vehículo. Esto tiene la ventaja de que los mismos parámetros de los controladores pueden servir para un vehículo u otro, permitiendo que si un usuario se construye su propio vehículo, este pueda más o menos estabilizarse con los valores por defecto de los parámetros. Sin embargo, no interesa ya que las ganancias de los controladores no tendrían significado físico. Por esta razón se ha eliminado esa saturación en el archivo *mixer_multirotor.cpp*, para así poder expresar los pares en la unidad $N \cdot m$ y la fuerza en N.

Otro cambio realizado, pero no a nivel de código, es la modificación de los parámetros del estimador EKF. En concreto, se han disminuido el ruido a asociado a los sensores ya que, como se comentó antes, se ha eliminado el ruido de la simulación de *Gazebo*. Estos parámetros se mostraron en la sección 2.4 y algunos son *EKF2_GPS_P_NOISE* y *EKF2_GYR_NOISE*. De esta manera los estados estimados serán muy paracidos a los reales como se va a ver en la siguiente sección.

5.4 Comparación con *Matlab/Simulink®*

Para verificar que este controlador funciona de la misma manera que el implementado en *Simulink* se ha realizado una comparación del seguimiento de velocidad angular en la figura 5.7 donde q_{gt} es el *groundtruth*, es decir, la medida real aportada por *Gazebo* a la cual no tienen acceso los controladores, y q_{est} es la medida estimada utilizada por los controladores. Los valores de la ganancias elegidos han sido los obtenidos en el diseño analítico del capítulo anterior, pero añadiéndole el término integral para comprobar que los mecanismos *antiwindup* funcionan de la misma manera.

Mirar las figuras 5.8, 5.9 y 5.10 donde se compara el seguimiento a ángulo, velocidad y posición respectivamente.

[5] Además, se debe de modificar el archivo de la carpeta Firmware/ROMFS/px4fmu_common/mixers/ que corresponda al vehículo elegido. De esta manera se indica que se haga uso del mixer creado

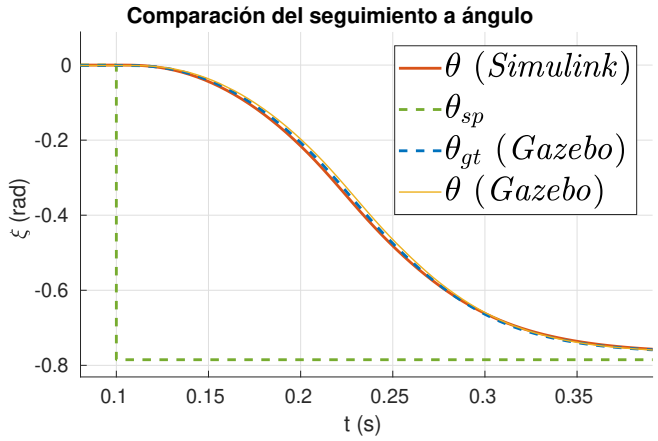

Figura 5.8 Comparación del seguimiento a ángulo.

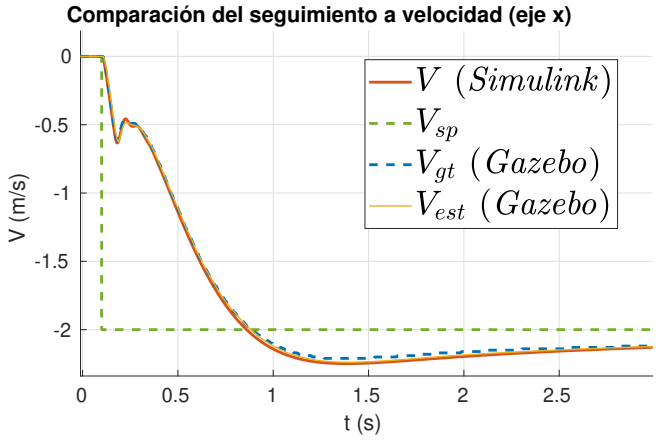

Figura 5.9 Comparación del seguimiento a velocidad. La ganancia integral es $K_i = 1\frac{N}{m}$.

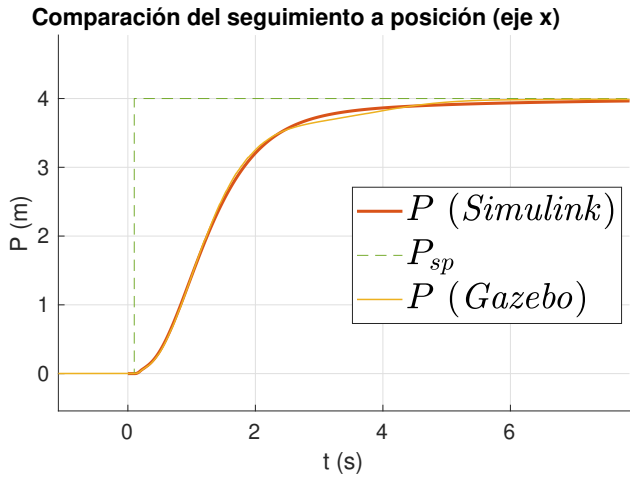

Figura 5.10 Comparación del seguimiento a posición.

6 Fabricación de un tiltrotor

En este capítulo se va a detallar los componentes elegidos en la construcción de un tiltrotor. El objetivo final es que sea capaz de seguir la orientación de referencia que se establece desde la emisora.

6.1 Componentes

- 1.x Componentes electrónicos

  En esta categoría se engloban los componentes electrónicos de baja potencia. Se muestran en la figura 6.1.

  - **1.1** Autopiloto *The Cube*. Este incorpora 2 microcontroladores, sensores como giróscopos, acelerómetros, magnetómetros... Algunas de sus especificaciones se describierón en la sección 2.1.
  - **1.2** GNSS *HERO*.
  - **1.3** Receptor *X8R*. Recibe hasta 16 canales de la emisora, que este caso es una *Taranis Q X7*.
  - **1.4** Módulo de telemetría *Holybro V3*. Permite una comunicación bidireccional con la estación de control terrestre.
  - **1.5** Zumbador. Emite señales auditivas para informar del estado del vehículo.

- 2.x Componentes de potencia
  - **2.1** *Tiger Motor MN4010 475KV*[1]. Motor sin escobillas. Hace girar una hélice de plástico con un diámetro de 14 pulgadas y un paso (o *pitch*) de 4.7 pulgadas. Según el fabricante del motor, ofrece un empuje de 1280 gramos alimentado a 14.8 V con una hélice de dichas características pero de fibra de carbono.
  - **2.2** *Hobbywing XRotor 40A*. Variador de velocidad o ESC.
  - **2.3** *Tattu Funfly 1300mAh*. Batería LiPo de 4 celdas.
  - **2.4** *Power Brick Mini*. Regulador de voltaje para alimentar el autopiloto. Además, lee el voltaje y la corriente que suministra la batería.
  - **2.5** *Hitec HS-5685MH*. Servomotor que puede aplicar un par de 12.9 Kg/cm (alimentado con 7.4V) a una velocidad de 0.17s/60°.
  - **2.6** *XLSEMI XL4015E1*. Reductor de voltaje capaz de suministrar 5A. Este se utilizará para alimentar los servomotores con 7.4V.

- 3.x Estructura

  La estructura está formada por barras de aluminio de perfil rectangular 10x12x1mm con una densidad lineal de $0.093\frac{kg}{m}$. Para unirlas se han usado tornillos y tuercas autoblocantes de rosca M3 y M4. La placa **3.5** es de metacrilato y tiene una densidad superficial de $3.71\frac{kg}{m^2}$.

[1] Cortesía del Departamento de Ingeniería de Sistemas y Automática de la Universidad de Sevilla

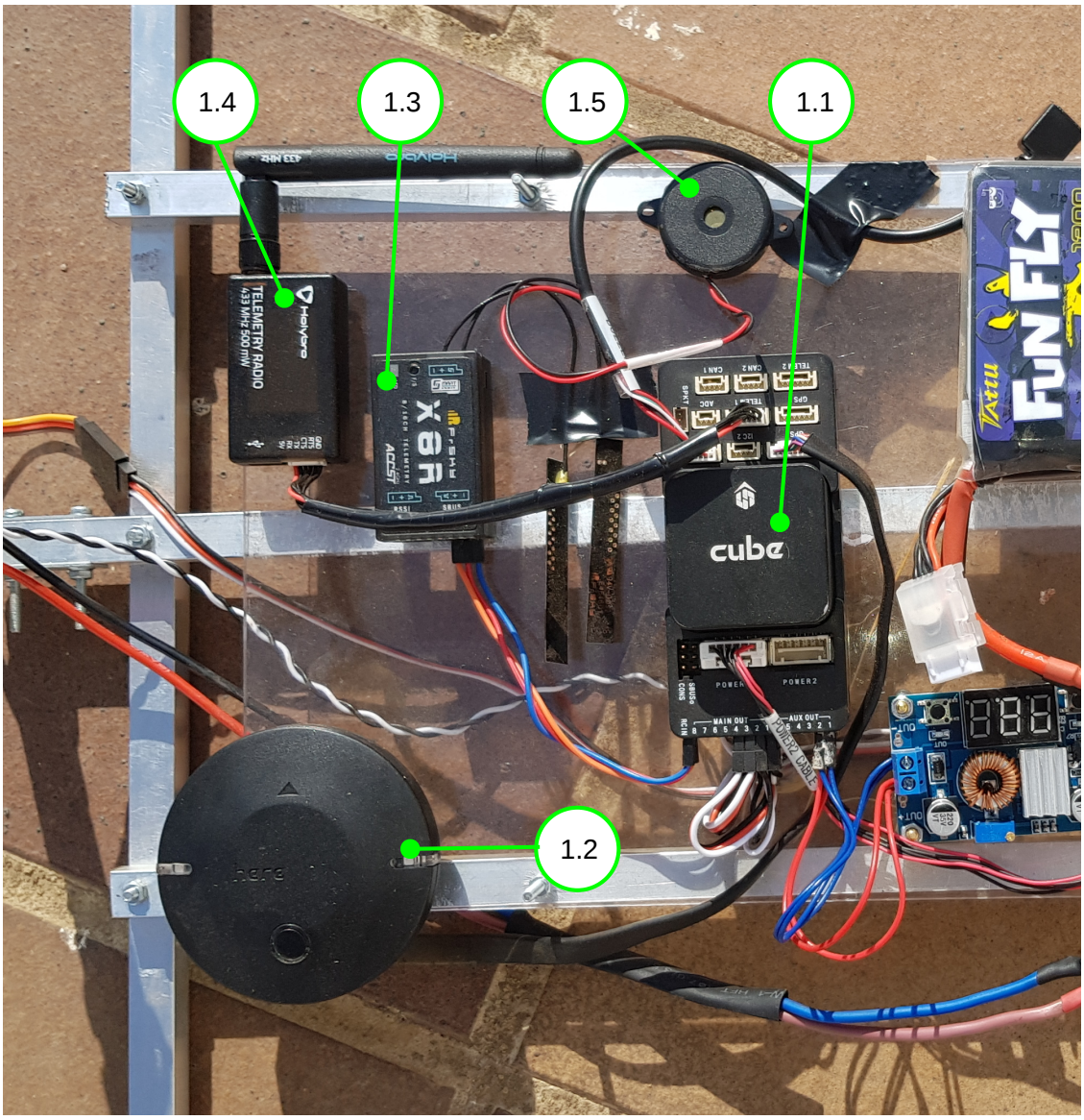

Figura 6.1 Componentes electrónicos.

- 4.x Articulación.

  Las articulaciones permiten que los motores que hacen girar las hélices puedan inclinarse. Su material es el acero y se muestra en detalle en la figura 6.3. Las piezas que restringen el movimiento al de una rotación son el tubo (**4.1**) y la varilla (**4.2**). Esta última se fija a la pieza en forma de U (**4.3**) mediante anillos de seguridad (circlip). De esa misma forma se fija el tubo a la pieza **4.4**, la cual tiene un perfil cuadrado y sirve para unir este conjunto a la barra **3.4**. Para transmitir el movimiento de la cabeza del servomotor a la articulación se ha usado unas rótulas (**4.5**).

6.2 Estimación de parámetros

Para empezar, se hallará la posición del centro de masas, para ello se conocen las masas y las posiciones de cada elemento expresadas en un sistema de referencia situado en el punto medio de la placa de metacrilato:

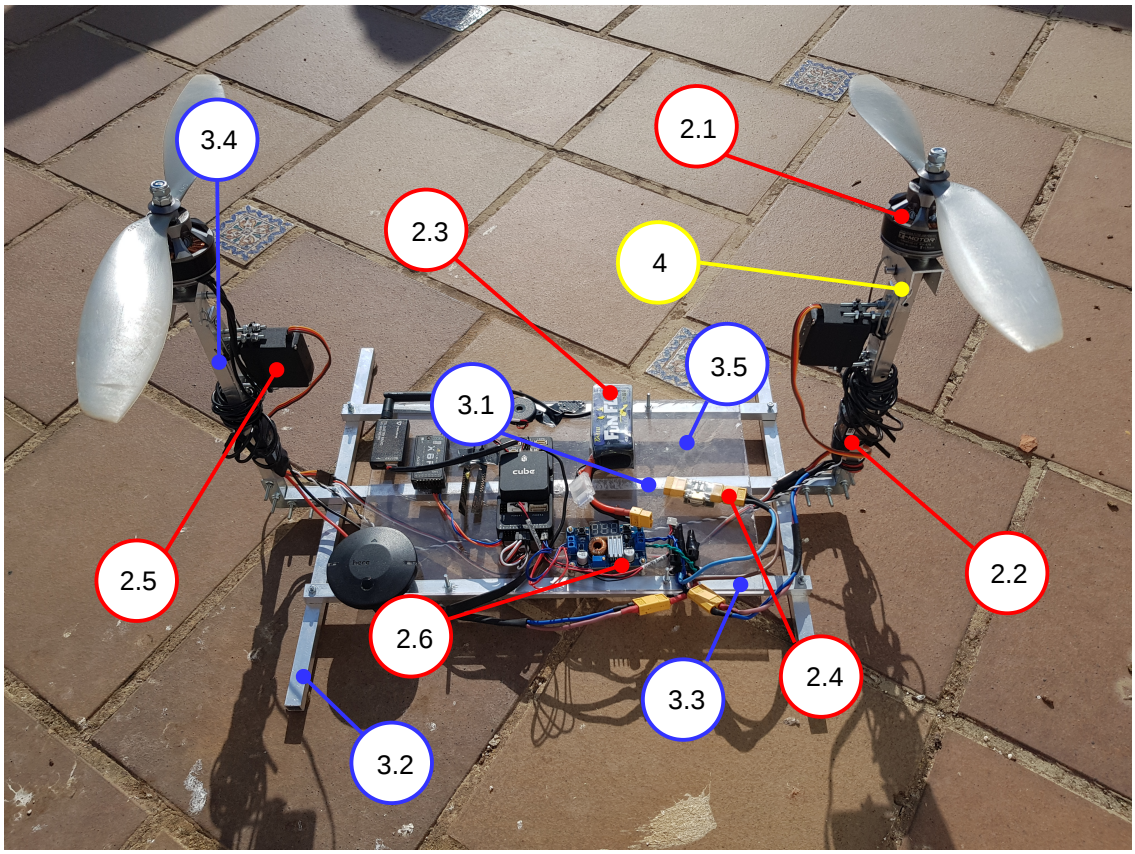

Figura 6.2 Componentes de potencia y de estructura.

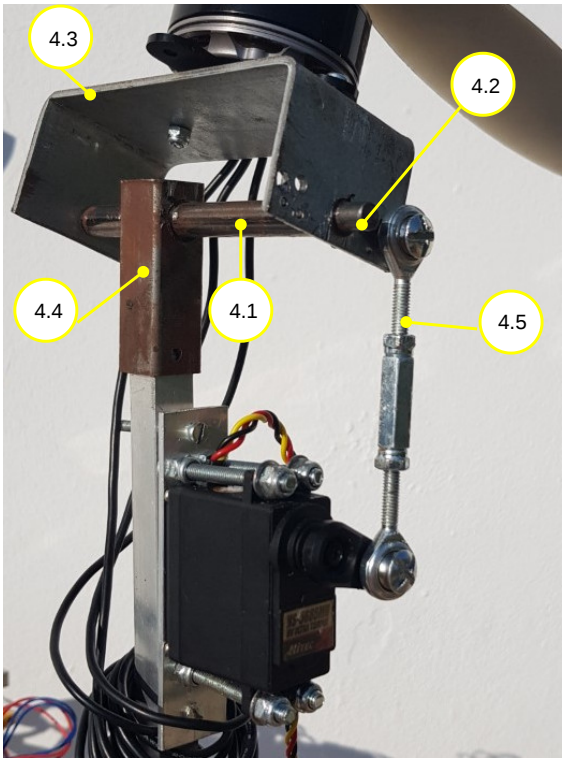

Figura 6.3 Partes de la articulación.

Elemento	Identificador	Posición (cm)			Masa (g)
		x	y	z	
Autopiloto	**1.1**	0	-2.5	0	120
GNSS	**1.2**	-9	-14	0	48
Receptor	**1.3**	3	-11	0	17
Telemetría	**1.4**	4	-14	0	24
Buzzer	**1.5**	9	-3	0	10
Motor + Hélice	**2.1a**	0	43	-30	149+35=184
Motor + Hélice	**2.1b**	0	-43	-30	149+35=184
ESC	**2.2a**	0	43	-7	26
ESC	**2.2b**	0	-43	-7	26
Batería	**2.3**	-	-	0	149
Módulo de alimentación	**2.4**	0	16	0	14
Servomotor	**2.5a**	0	45	-17	57
Servomotor	**2.5b**	0	-45	-17	57
Regulador	**2.6**	-5	4	0	18
Aluminio 85cm[2]	**3.1**	0	0	0	$0.85\,m \cdot 93\frac{g}{m} = 79$
Aluminio 35cm	**3.2a**	0	20	0	$0.35\,m \cdot 93\frac{g}{m} = 32.6$
Aluminio 35cm	**3.2b**	0	-20	0	$0.35\,m \cdot 93\frac{g}{m} = 32.6$
Aluminio 39cm	**3.3a**	9	0	0	$0.39\,m \cdot 93\frac{g}{m} = 36.3$
Aluminio 39cm	**3.3b**	-9	0	0	$0.39\,m \cdot 93\frac{g}{m} = 36.3$
Aluminio 25cm	**3.4a**	0	43	0	$0.25\,m \cdot 93\frac{g}{m} = 23.3$
Aluminio 25cm	**3.4b**	0	-43	0	$0.25\,m \cdot 93\frac{g}{m} = 23.3$
Metacrilato 19x32cm	**3.5**	0	0	0	$0.19\,m \cdot 0.32\,m \cdot 3.71\frac{kg}{m^2} = 226$
Articulación	**4a**	0	43	-25	148
Articulación	**4b**	0	-43	-25	148

Tabla 6.1 Características de cada componente del tiltrotor.

Se calcula el centro de masas sin la batería (P_{aux}) mediante el siguiente sumatorio:

$$P_{aux} = \frac{1}{m_{aux}} \Sigma m_i\, P_i = \begin{bmatrix} -0.2108 \\ -0.7755 \\ -13.2503 \end{bmatrix} \tag{6.1}$$

Interesa que el centro de masas no tenga componente x ni y para aproximar este vehículo al modelo utilizado en los capítulos anteriores. Para conseguirlo, se hallará la posición de la batería que cumple dicha condición:

$$P_{cm} = \begin{bmatrix} 0 \\ 0 \\ z_{cm} = \end{bmatrix} = \frac{1}{m_T}\left(P_{aux} m_{aux} + P_{bat} m_{bat}\right) = \frac{1}{m_T}\left(\begin{bmatrix} x_{aux} \\ y_{aux} \\ z_{aux} \end{bmatrix} m_{aux} + \begin{bmatrix} x_{bat} \\ y_{bat} \\ 0 \end{bmatrix} m_{bat}\right) \tag{6.2}$$

Se despeja la posición de la batería:

$$x_{bat} = -\frac{m_{aux}}{m_{bat}} x_{aux} = -\frac{1565}{149} - .0.2108cm = 2.2148cm \tag{6.3}$$

$$y_{bat} = -\frac{m_{aux}}{m_{bat}} y_{aux} = -\frac{1565}{149} - 0.7755cm = 8.1477cm \tag{6.4}$$

Finalmente la posición del centro de masas queda:

$$P_{cm} = \begin{bmatrix} 0 \\ 0 \\ -12.0987 \end{bmatrix} \tag{6.5}$$

[2] La longitud de esta barra es mayor que la que aperece en la figura 6.2, ya que posteriormente el tiltrotor se modificó sustituyendo esa barra por una más larga.

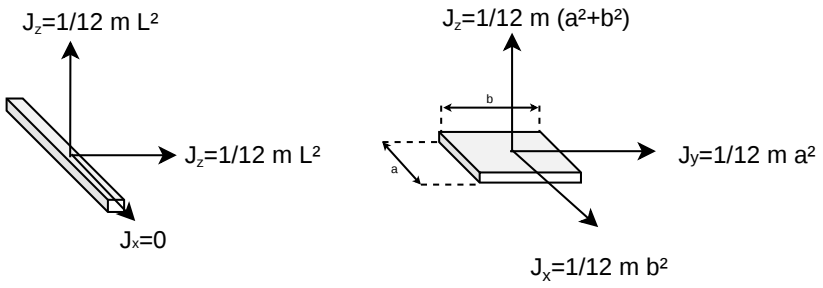

Figura 6.4 Cálculo de las inercias.

Una vez que se tiene la posición del centro de masas ya se puede calcular la inercia total del vehículo. Para su cálculo, se considerará que todos los elementos son objetos puntuales (la inercia medida en su centro de masas es nula) menos las barras de aluminio y la placa de metacrilato. Se hallará la inercia en unos ejes cuyo origen sea su centro de masas y paralelos a los ejes principales del vehículo. Utilizando las fórmulas de la figura 6.4 se halla:

Elemento	J_x	J_y	J_z
3.1	$\frac{1}{12}\cdot 0.079\cdot 85^2 = 44Kg\cdot cm^2$	0	$44Kg\cdot cm^2$
3.2	0	$\frac{1}{12}\cdot 0.033\cdot 35^2 = 3.37Kg\cdot cm^2$	$3.37Kg\cdot cm^2$
3.3	$\frac{1}{12}\cdot 0.036\cdot 39^2 = 4.56Kg\cdot cm^2$	0	$4.56Kg\cdot cm^2$
3.4	$\frac{1}{12}\cdot 0.023\cdot 25^2 = 1.20Kg\cdot cm^2$	$1.20Kg\cdot cm^2$	0
3.5	$\frac{1}{12}\cdot 0.226\cdot 32^2 = 19.29Kg\cdot cm^2$	$\frac{1}{12}\cdot 0.226\cdot 19^2 = 6.80Kg\cdot cm^2$	$19.29+6.80 = 26.09Kg\cdot cm^2$

Tabla 6.2 Inercias obtenidas de las barras y la placa.

Por último, se halla la inercia total medida en el centro de masas del vehículo. Se utilizará el teorema de Steiner generalizado a tensores de inercia:

$$P_i^{cm} = P_i - P_{cm} \tag{6.6}$$

$$J_i^{cm} = J_i + m_i\left(\|P_i^{cm}\|^2 I_{3x3} - P_i^{cm}\otimes P_i^{cm}\right) \tag{6.7}$$

La inercia total se calcula como la suma de la inercia de cada elemento medida en el centro de masas del vehículo:

$$J_{cm} = \Sigma J_i^{cm} = \begin{bmatrix} 2076 & -6 & 0 \\ -6 & 328 & 0 \\ 0 & 0 & 1795 \end{bmatrix} \quad [Kg\cdot cm^2] \tag{6.8}$$

Estos resultados no serán utilizados para diseñar los controladores en este trabajo.

6.3 Resultados

El autopiloto posee una tarjeta sd donde se guarda el historial o *log*, el cual almacena datos útiles para el análisis después del vuelo. Estos tienen formato *ulog* y existen varias herramientas que representan graficamente su contenido. De todas ellas, se ha elegido una herramienta online que está ubicada en review.px4.io.

En las figuras 6.6 y 6.7 se muestra el seguimiento a los ángulos roll y pitch en un fragmento de vuelo de 6 segundos en el modo *stabilized*.

En la figura 6.8 se puede observar las cuatro señales de actuación, medidas en microsegundos del ancho de pulso PWM. Los actuadores 0 y 1 son los ESCs de la derecha y de la izquierda vistos desde el punto de vista de un supuesto piloto. El actuador 2 es el servomotor de la derecha y el 3 es el de la izquierda. Se puede apreciar que existe una diferencia en la media de estos dos últimos de unos $200us$. Esto se debe a que se ha tenido que aplicar un offset diferente a cada uno de ellos, para que se encuentren en vertical cuando el par deseado en los ejes x e y sea 0.

Por último, en la figura 6.9 se muestra el nivel de las palancas de la emisora normalizados al rango [-1,1], exepto el acelerador que comienza en 0.

Figura 6.5 Tiltrotor en pleno vuelo.

Figura 6.6 Seguimiento del ángulo roll.

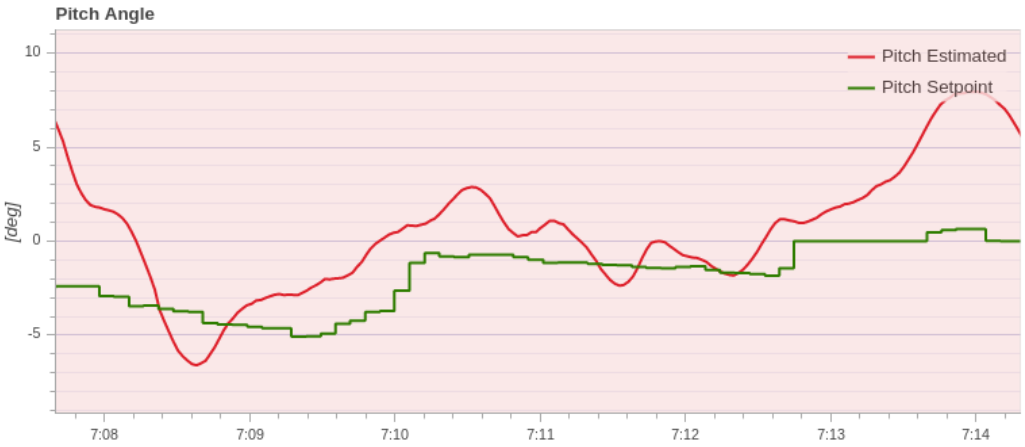

Figura 6.7 Seguimiento del ángulo pitch.

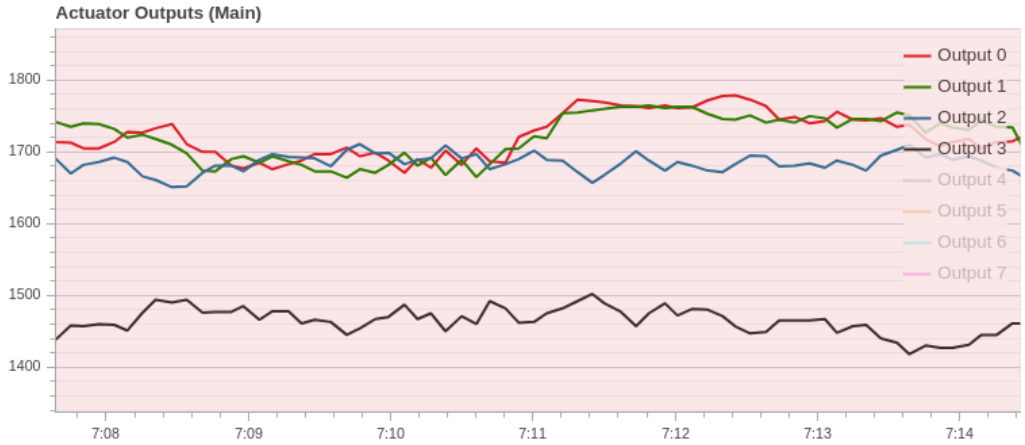

Figura 6.8 Señales de actuación.

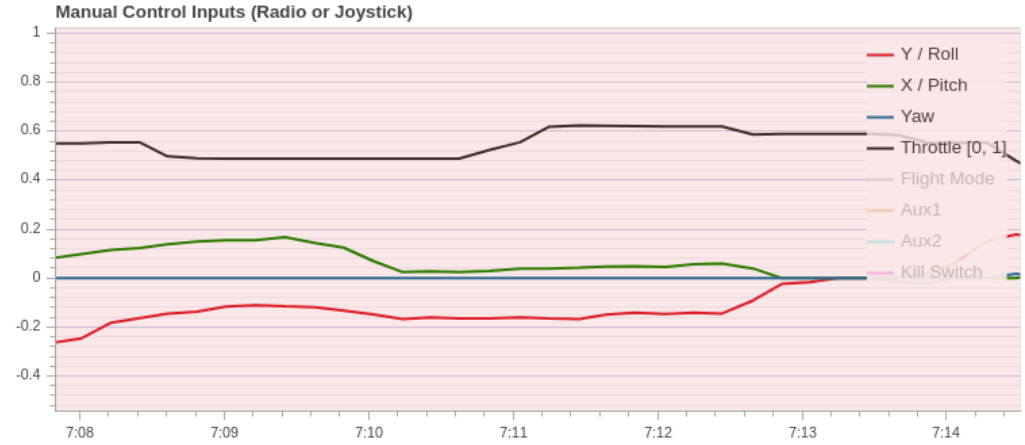

Figura 6.9 Posición de las palancas del mando.

7 Conclusiones

Se ha podido ver que el **autopiloto** *PX4* ofrece una gran cantidad de funcionalidades, que vienen validadas por la multitud de vuelos que se realizan con este autopiloto. Además, la simulación y la depuración que ofrece, es una forma ideal para agregarle cambios sin que el vehículo real esté en peligro. Por otro lado, hay que tener en cuenta que *PX4* tiene una gran flexibilidad ya que admite varias configuraciones de vehículos y bastantes combinaciones de componentes. Quizás esto provoque que sea un poco más complejo y menos óptimo que en el caso en el que se cree un autopiloto para un vehículo en concreto. Aun así, a la hora de crear un vehículo aéreo, resulta conveniente estudiar estos tipos de trabajos. Ya que entender lo que otros han hecho antes y tomarlo como punto de partida para crear proyectos propios y novedosos, permite que el avance de la comunidad investigadora sea más rápido que en el caso de que cada uno empiece de cero y no comparta sus descubrimientos.

En cuanto al **diseño de controladores** y **simulación** se ha podido ver que el controlador que incorpora el autopiloto es aplicable a un tiltrotor haciendo unas pocas modificaciones. En el diseño analítico, las simplificaciones que se han tomado sobre el modelo hacen que las especificaciones no se cumplan, pero aun así resulte un comportamiento estable. En cuanto al diseño iterativo se ha comprobado que no tiene por qué ofrecer mejores resultados que el analítico si no se eligen las funciones de coste adecuadas.

Por último, en la **fabricación** se ha visto una de las grandes ventajas de utilizar *PX4*: pasar de la simulación a una implementación real consiste únicamente en comprar los componentes y conectarlos, prácticamente no hay que modificar nada de software. También se ha comprobado que la estructura del controlador utilizado en la simulación puede aplicarse a un modelo real, a pesar de sus diferencias con respecto al de la simulación.

8 Trabajos futuros

Los trabajos futuros vendrán marcados por la importante experiencia que me ha supuesto el desarrollo de este proyecto, tanto en la investigación y estudio de trabajos ya realizados, como en mi experimentación de prueba y error que me han ayudado en mi objetivo de llevar a buen puerto este Trabajo de Fin de Grado. A continuación, relaciono una serie de elementos y acciones en los que me gustaría profundizar en futuros proyectos:

Autopiloto:

- Ampliar los archívos históricos (*logs*) con variables útiles como la estimación de la aceleración angular o cada uno de los términos de los PID.
- Añadir offsets a la salida de los controladores como, por ejemplo, en la orientación deseada. Puesto que para que el vehículo permanezca estático no tiene por qué estar en horizontal. [1].

Diseño de controladores y simulación:

- Unificar los parametros de *Matlab*, *Gazebo*, *PX4*; de manera que al cambiar los parámetros físicos del vehículo (distancias, constantes de tiempo, ...) solo haya que modificar un único archivo.
- Utilizar otras funciones de coste para la optimización, teniendo en cuenta, por ejemplo, la sobre-oscilación, la energía en el dominio frecuencial o las señales de actuación.

Fabricación:

- Utilizar servomotores con realimentación de posición y añadirla a los *logs*. También se podría ir un paso más allá y utilizar servomotores inteligentes los cuales además indican la velocidad, corriente y voltaje.
- Fabricar una articulación con rodamientos y evitar el uso del acero.
- Identificar todos los parámetros del vehículo real: dinámica de los motores sin escobillas, dinámica de los servomotores, empuje y par que generan las hélices ...
- Utilizar fibra de carbono.
- Comprobar si se pueden utilizar servomotores de menor par, ya que estos son más rápidos sin tener que ser necesariamente más costosos (relación de trasmisión más alta).
- Utilizar controladores que no asuman tantas simplificaciones.
- Añadir superficies aerodinámicas para hacer viable el modo avión.

[1] Esto es causado porque, para que el vehículo no rote, los rotores tienen que estar inclinados de tal manera que la dirección de empuje se alinee con el centro de masas. Si el centro de masas, tiene una componente en el eje x no nula, al contrario de lo considerado en este trabajo, la única manera de que el vehículo permanezca estático es inclinándose

Apéndice A
Archivo de parámetros

Código A.0.1: Script donde se asignan todos los parámetros (*parametros.m*)

```
global param

%%% Modelo tiltrotor %%%
param.m=1; % Masa total del tiltrotor [Kg]
param.lz=0.25; % Distancia en el eje z desde el c.m. hasta los rotores
param.ly=0.5; % Distancia en el eje y desde el c.m. hasta los rotores
param.Jx=0.03; % Inercia en el eje x [Kg.m^2]
param.Jy=0.04; % Inercia en el eje y [Kg.m^2]
param.Jz=0.09; % Inercia en el eje z [Kg.m^2]
param.g=9.8; % Aceleración de la gravedad [m/s^2]
param.cT=8.0992e-06; % Coeficiente de empuje de los rotores
param.cm=0; % Coeficiente de arrastre de los rotores
param.max_omega=1100; % Máxima velocidad de los rotores [rad/s]
param.max_alfa=pi/3; % Máxima inclinación de lor rorores [rad]
param.max_taux=30; % Máximo par en el eje x que genera control de velocidad angular [N.m]
param.max_tauy=30; % Máximo par en el eje y que genera control de velocidad angular [N.m]
param.max_tauz=30; % Máximo par en el eje z que genera control de velocidad angular [N.m]
param.max_Fz=2*(param.max_omega)^2*param.cT; % Máxima fuerza en el eje z del cuerpo  [N]
%param.max_T=2*(max_omega)^2*cT; %Hay dos rotores
param.tau_omega=0.01; % Constante de tiempo de los motores que hacen girar los rotores [s]
param.tau_alfa=0.01; % Constante de tiempo de los motores que inclinan el conjunto motor-rotor [s]

%%% Opciones de la simulación %%%
param.modoControl=6; % Esta variable indica cuales de los controladores están activos. 0- Ningún controlador; 1 - Control Mixer; 2 - Control de velocidad angular; 3 - Control de ángulo; 4 - Control de Fuerzas en ejes inerciales; 5 - Control de velocidad; 6 . Control de posición.
param.Tm=4e-3; % Tiempo de discretización de algunos controladores [s]
param.tsim_run_simu=8; % tiempo de simulación [s].
param.step_ref=0.1; % Instante de tiempo en el que se aplica el escalon a la referencia [s]

%%% Referencias %%%
param.omega1_ref=1100; % Velocidad de giro del rotor 1 [rad/s]. Tiene efecto cuando modoControl=0.
param.omega2_ref=1100; % Velocidad de giro del rotor 2 [rad/s]. Tiene efecto cuando modoControl=0.
param.alfa1_ref=0; % Ángulo de inclinación del motor 1 [rad]. Tiene efecto cuando modoControl=0.
param.alfa2_ref=0; % Ángulo de inclinación del motor 1 [rad]. Tiene efecto cuando modoControl=0.
param.tau_ref=[0;0;0]; % Referencias de par  en los tres ejes [N.m] (modoControl=1).
param.Fz_ref=param.g*param.m; % Fuerza en el eje z del vehículo [N] (modoControl=1 || modoControl=2).
```

```
param.omega_ref=[0;0;0]; % Referencias de velocidades angulares en los tres ejes [rad/s] (modoControl=2).
param.eul_ref=[0,0,0]; % Referencias de orientación expresadas en ángulor de euler en el orden roll, pitch y yaw [rad] (modo_control=3).
param.FNED_ref=[param.g*param.m/4;0;-param.g*param.m]; % Referencias de fuerza expresada en ejes inerciales [N] (modoContros=4).
param.yaw_sp=0; % Ángulo yaw deseado [rad] (modoControl=4).
param.v_ref=[0;0;0]; % Velocidad de referencia [m/s] (modoControl=5).
param.p_ref=[0;0;0]; % Posición de referencia [m] (modoControl=6).
param.eje_exp=1; % indica en qué eje del vehículo se hará el experimento: 0->eje x; 1->eje y; 2->eje z
amp_omega=3.5; % Amplitud del escalón en la referencia de velocidad angular [rad/s]. Será aplicado en un eje u otro dependiendo del valor de 'eje_exp'
amp_v=-2; % Amplitud del escalón en la referencia de velocidad [m/s]
amp_p=4; % Amplitud del escalón en la referencia de posición [m]
amp_tau=2.5; % Amplitud del escalón en la referencia de  [m]
amp_eul=-0.785; % Amplitud del escalón en la referencia de posición [m]
switch param.eje_exp
    case 0
        % Eje x
        param.omega_ref(1)=amp_omega;
        param.tau_ref(1)=amp_tau;
        param.eul_ref(1)=amp_eul;
        param.v_ref(2)=amp_v; % Si se está evaluando el control de rotación sobre el eje x, habrá que generar un escalón de velocidad en el eje y
        param.p_ref(2)=amp_omega;
    case 1
        % Eje y
        param.omega_ref(2)=amp_omega;
        param.tau_ref(2)=amp_tau;
        param.eul_ref(2)=amp_eul;
        param.v_ref(1)=amp_v;
        param.p_ref(1)=amp_p;
    case 2
        % Eje z
        param.omega_ref(3)=amp_omega;
        param.tau_ref(3)=amp_tau;
        param.eul_ref(3)=amp_eul;
        param.v_ref(3)=amp_v;
        param.p_ref(3)=amp_p;
end
param.q_ref=eul2quat(fliplr(param.eul_ref)); % Referencia de orientación expresada en cuaternios. La función 'eul2quat' recibe los ángulos de euler en orden yaw, pitch y yaw, por esa razón se utiliza la otra función 'fliplr' para invertir el orden del vector.

%%% Perturbaciones %%%
param.F_pert=[0;0;0]; % Fuerza de perturbación en ejes inerciales [N].
param.tau_pert=[0;0;0]; % Pares de perturbación en ejes cuerpo [N.m]
amp_tau_pert= 0.5; % N.m
amp_F_pert=0.2; % N
if param.modoControl==2 || param.modoControl==3
    % Solo se aplicará una perturbación de par cuando se estén evaluando el controlador de velocidad angular o el de ángulo
    switch param.eje_exp
    case 0
        param.tau_pert(1)=amp_tau_pert;
    case 1
        param.tau_pert(2)=amp_tau_pert;
    case 2
        param.tau_pert(3)=amp_tau_pert;
    end
end
if param.modoControl==5 || param.modoControl==6
    % Solo se aplicará una perturbación de fuerza cuando se estén evaluando o el controlador de velocidad o el de posición.
    switch param.eje_exp
    case 0
        param.F_pert(2)=amp_F_pert;
```

```
94      case 1
95          param.F_pert(1)=amp_F_pert;
96      case 2
97          param.F_pert(3)=amp_F_pert;
98      end
99  end
100 param.delay_pert=param.tsim_run_simu/2; % Instante en el que se aplica la perturbación
      ↪  mantenida [s].
101
102 %%% Controlador de posición %%%
103 param.MPC_XY_P=0.6981;
104 param.MPC_Z_P=0.5;
105
106 %%% Controlador de velocidad %%%
107 param.MPC_Z_VEL_P=39.4784;
108 param.MPC_Z_VEL_I=0;
109 param.MPC_Z_VEL_D=0.2566;
110 % Analítico
111 param.MPC_XY_VEL_P=  2.9588 ;
112 param.MPC_XY_VEL_I=0;
113 param.MPC_XY_VEL_D=  0.0472;
114 % optimización
115 % param.MPC_XY_VEL_P= 7.3701;
116 % param.MPC_XY_VEL_I= 0.4456;
117 % param.MPC_XY_VEL_D= 0.1202;
118 param.MPC_THR_HOVER=param.g*param.m;
119 param.MPC_THR_MAX=param.g*param.m*2;
120 param.MPC_THR_MIN=0;
121 param.MPC_TILTMAX_AIR=45; %grados
122 param.MPC_XY_VEL_MAX=12;
123
124 %%% Controlador de orientación  %%%
125 param.MC_ROLL_P=12.57;
126 param.MC_PITCH_P=10.4720; % analítico
127 % param.MC_PITCH_P=12.4518; %optimización
128 param.MC_YAW_P=12.57;
129 param.MC_ROLLRATE_MAX=200;
130 param.MC_PITCHRATE_MAX=360;
131 param.MC_YAWRATE_MAX=200;
132
133 %%% Controlador de velocidad angular %%%
134 % Diseño analitico
135 param.MC_ROLLRATE_P=0;
136 param.MC_ROLLRATE_I=0;
137 param.MC_ROLLRATE_D=0;
138 % analítico
139 param.MC_PITCHRATE_P=2.4674;
140 param.MC_PITCHRATE_I=0;
141 param.MC_PITCHRATE_D=0.0228;
142 % optimización
143 % param.MC_PITCHRATE_P=4.2473;
144 % param.MC_PITCHRATE_I=22.5534;
145 % param.MC_PITCHRATE_D=0.0343;
146 param.MC_YAWRATE_P=0;
147 param.MC_YAWRATE_I=0;
148 param.MC_YAWRATE_D=0;
149 param.MC_RR_INT_LIM=1;
150 param.MC_PR_INT_LIM=0.5;
151 param.MC_YR_INT_LIM=1;
152
153
154 %%% Filtros %%%
155 % Coeficientes del filtro para estimar la aceleración angular
156 param.MC_DTERM_CUTOFF=30;
157 K=tan(2*pi*30*4e-3/2);
158 DE=K^2 + sqrt(2)*K +1;
159 param.b0=K^2/DE;
160 param.b1=2*K^2/DE;
161 param.b2=K^2/DE;
```

```
param.a1=2*(K^2 -1)/DE;
param.a2=(1-K*sqrt(2) + K^2)/DE;
% Filtro utilizado para estimar la aceleración lineal
% param.MPC_VELD_LP=5;
 param.MPC_VELD_LP=5;
b=2*pi*param.MPC_VELD_LP*param.Tm*2;
param.a=b/(1+b);

clearvars -except param
```

Índice de Figuras

Índice de Tablas

Bibliografía

[1] *Telefónica realiza con éxito un piloto con drones basado en soluciones IoT para la detección temprana de incendios*. 2019. URL: *https://www.telefonica.com/web/sala-de-prensa/-/telefonica-realiza-con-exito-un-piloto-con-drones-basado* (visitado 05-07-2019).

[2] Andrew Liptak. *The FAA says the commercial drone market could triple in size by 2023*. 2019. URL: *https://www.theverge.com/2019/5/4/18529241/faa-annual-aviation-report-hobby-commercial-drones-prediction-2023* (visitado 05-07-2019).

[3] Zhong Liu y col. «Control techniques of tilt rotor unmanned aerial vehicle systems: A review». En: *Chinese Journal of Aeronautics* 30.1 (2017), págs. 135-148. ISSN: 1000-9361. DOI: *10.1016/j.cja.2016.11.001*. URL: *http://www.sciencedirect.com/science/article/pii/S1000936116302199*.

[4] A. Sanchez y col. «Autonomous Hovering of a Noncyclic Tiltrotor UAV: Modeling, Control and Implementation». En: *IFAC Proceedings Volumes* 41.2 (2008). 17th IFAC World Congress, págs. 803-808. ISSN: 1474-6670. DOI: *10.3182/20080706-5-KR-1001.00138*. URL: *http://www.sciencedirect.com/science/article/pii/S1474667016390486*.

[5] R. Donadel, G.V. Raffo y L.B. Becker. «Modeling and Control of a Tiltrotor UAV for Path Tracking». En: *IFAC Proceedings Volumes* 47.3 (2014). 19th IFAC World Congress, págs. 3839-3844. ISSN: 1474-6670. DOI: *10.3182/20140824-6-ZA-1003.01735*. URL: *http://www.sciencedirect.com/science/article/pii/S1474667016422020*.

[6] Dario Brescianini, Markus Hehn y Raffaello DÁndrea. *Nonlinear Quadrocopter Attitude Control. Technical Report*. en. Inf. téc. 2013. DOI: *10.3929/ethz-a-009970340*.

[7] Randal W. Beard. *Quadrotor dynamics and control*. Inf. téc. 2008. URL: *http://hdl.lib.byu.edu/1877/624*.

[8] Randal W Beard y Timothy W McLain. *Small unmanned aircraft: Theory and practice*. Princeton university press, 2012.

[9] A. Q. Ansari, Ibraheem y S. Katiyar. «Application of ant colony algorithm for calculation and analysis of performance indices for adaptive control system». En: (nov. de 2014), págs. 466-471. DOI: *10.1109/CIPECH.2014.7019078*.